Nastaleeq: A challenge accepted by Omega

Atif Gulzar, Shafiq ur Rahman
Center for Research in Urdu Language Processing,
National University of Computer and Emerging Sciences, Lahore, Pakistan
`atif dot gulzar (at) gmail dot com, shafiq dot rahman (at) nu dot edu dot pk`

Abstract

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

With the advent of multilingual support in computer systems, different solutions have been proposed and implemented. But most of these are immature or platform-specific. This paper discuses the complexity of Nastaleeq and a solution that uses Omega as the typesetting engine for rendering Nastaleeq.

1 Introduction

Urdu is the lingua franca as well as the 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. The Nastaleeq writing style was developed by Mir Ali Tabrazi in 14th century by combining Naskh and Taleeq (an old obsolete style) [2].

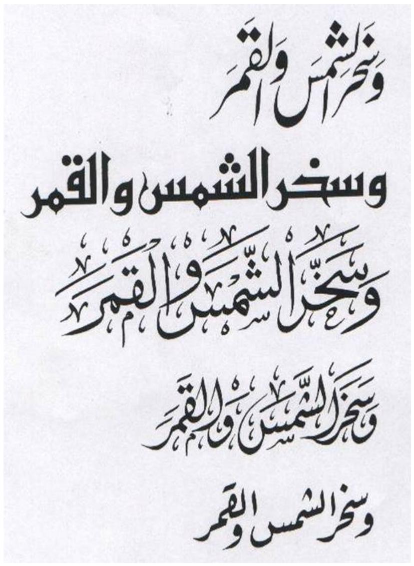


Figure 1: Different Arabic writing styles (from top to bottom: Nastaleeq, Kufi, Sulus, Deevani and Riqah) [3]

1.1 Complexity of the Nastaleeq writing style

The Nastaleeq writing style is far more complex than other writing styles of Arabic script–based languages. The salient features‘r of Nastaleeq that make it more complex are these:

- Nastaleeq is a cursive writing style, like other Arabic styles, but it is written diagonally from right-to-left and top-to-bottom, as shown in figure 2. Numerals add to the complexity as they are written from left-to-right (figure 7).

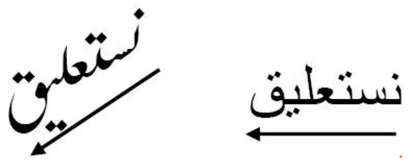


Figure 2: Direction of Nastaleeq writing style

- In most Arabic styles (especially digitized forms (fonts) of these styles), each character may assume up to four different shapes (isolated, initial, medial and final) depending on its position in the ligature. The character *Beh* (U+0628) takes four shapes depending on its position in isolated (a), initial (b), medial (c) or 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 only dependent on its position in a ligature but also on the shapes of the neighboring characters (mostly on the shape of the character that

جب	جبا	با	ب
d	c	b	a

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

بی	بط	بج	با
جبط	جبپ	بر	جبے

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

follows it). Table 2 shows a subset of the variations of *Beh* in different contexts. In Nastaleeq a single character may assume up to 50 shapes.

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

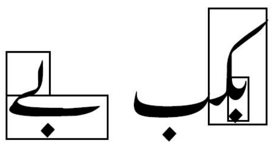
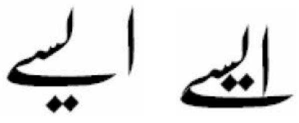


Figure 3: Overlapping glyphs in Nastaleeq

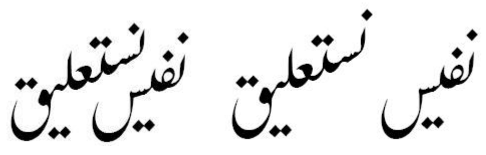
These overlapping shapes in Nastaleeq pose a major concern for kerning, proportional spacing and nuqta placement. As shown in figure 4, the ligature needs to be kerned to avoid clashing with the preceding ligature:



(b) (a)

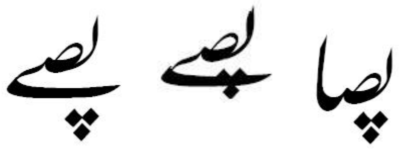
Figure 4: before (a) and after (b) kerning

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



(b) (a)

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



(c) (b) (a)

Figure 6: (a) nuqtas at default position;
(b) default nuqta positioning produces a clash in different contexts; (c) default nuqtas are repositioned contextually to avoid clash.

1.2 Current Solutions

Two different techniques have been adopted for digitizing the Nastaleeq script: a ligature-based approach and a character-based approach. Each has its own limitations. The most dominant and widely used solution is the ligature-based Nori Nastaleeq. It has over 20,000 pre-composed ligatures [2]. This font can only be used with the proprietary software InPage. The other promising solutions are character-based OpenType fonts. These fonts use OpenType technology to generate ligatures. The OpenType solution is very slow for the 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 in handling 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 the Windows platform. The support currently 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 the Nastaleeq script. There is a need to provide a platform-independent solution for Nastaleeq.

The solution devised here provides Nastaleeq rendering support in Linux through Omega. Omega has the strong underlying typesetting system TEX to handle the complexity of Nastaleeq rendering and

Omega Translation Processes (ΩTPs) provide a solution for the complexity of Nastaleeq script (e.g. contextual shape substitution) [4].

The present solution is limited to the basic alphabets of Urdu (ا (U+0627) to ے (U+06D2)) and numerals (0 to 9). These alphabets are listed in Appendix A. The solution provides:

- correct glyph substitution according to the contextual dependency of a character.
- correct cursive attachment(s) of a glyph
- nuqta placement
- automatic bidirectional support for numerals

2 Methodology

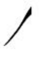
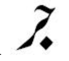
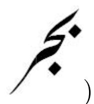
There are two possibilities for implementing support for Nastaleeq in Omega: internal ΩTPs and external ΩTPs. It is observed that internal ΩTPs are syntax dependent; for example, it is almost impossible to implement reverse chaining (processing characters/glyphs in the reverse order in a ligature) using the syntax of internal ΩTPs. External ΩTPs can be implemented using Perl or C/C++, and give the 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 sections discuss 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 An Omega virtual font for Nastaleeq

An Omega virtual font file is generated from a Nafees Nastaleeq TTF font file. A total of 827 glyphs have been used to render Nastaleeq. These glyphs are placed in four different Type 1 files and four different TFM files are also generated. The Omega program itself uses only the single virtual font file `nafees.ofm` that contains pointers to the above generated font files.

2.2 Substitution logic

Nastaleeq 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 the character that follows it. However, the shape of a final character in a ligature is dependent on the second to last character, with a few exceptions. For example, the character *Reh* (ر, U+0631) has two glyphs and  (as in  and ) when the character *Jeem* (ج, U+062C) occurs at the initial and medial position of a ligature, respectively. Similarly, characters U+0631, U+0691, U+0632, U+0698, U+0642, U+0648 and U+06CC all have different final glyphs depending on the glyph of the 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 shown in Table 3 below.

	U+0628	U+0629	U+0630
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 the table consists of Unicode values. The remainder are 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 the final character two final tables are used: *final1* and *final2* for two character combinations and more than two character combinations, respectively. It is needed because final shape depends on the rightmost character; and there are only two possibilities for a character at the $(n-1)^{\text{th}}$ position: either it is an initial shape (in a two character combination) or a medial shape (in a more than two character combination).

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

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

Table 4: Format of lookup table for final shape context

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

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

For the remaining $n-2$ characters, the medial table is consulted. The shape of the n^{th} character can be found in the medial table as follows:

```
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 found 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 to see if it is composed of numerals. In case of numerals, the string is printed in reverse order, so as 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]
```

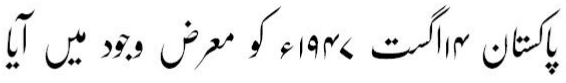

پاکستان ۱۴ اگست ۱۹۴۷ء کو معرض وجود میں آیا

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; instead, the baseline of each glyph is dependent on the 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 anything about the shape of a character. It only knows the box with height, width and depth properties. TeX output contains a list of boxes concatenated with each other. By default these boxes are aligned along the baseline (Fig. 8). But these boxes can be shifted horizontally or vertically.

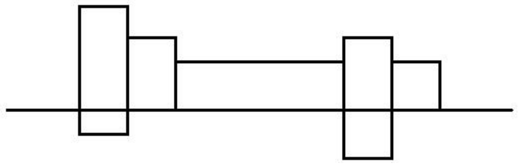


Figure 8: TeX boxes

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

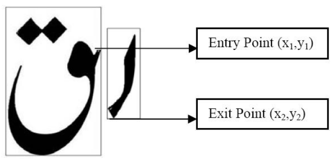


Figure 9: Entry and exit points

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

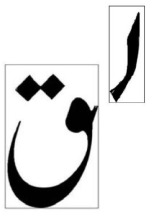


Figure 10: After vertical adjustment

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

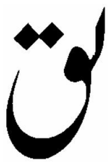


Figure 11: After vertical and horizontal adjustment

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 this way because the n^{th} (last) glyph of a ligature always resides on the baseline, while the other $n-1$ glyphs move vertically upward according to the entry and 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 the *enex* table contains the entry and exit points, the *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 the other creates a clash between ligatures (b). Case (a) is not included in the present implementation, but case (b) is handled.

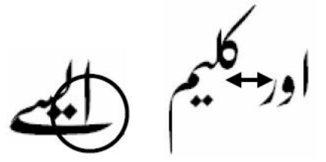


Figure 12: Types of kerning problems

The final shapes of the characters *Yeh Barree* (ے, U+06D2), *Jeem* (ج, U+062C) and *Ain* (ع, U+0639) produce (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 final glyph from the sum of widths of the 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 cannot be placed at fixed positions, but must 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:

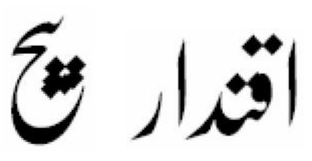


Figure 13: Nuqta collision types

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

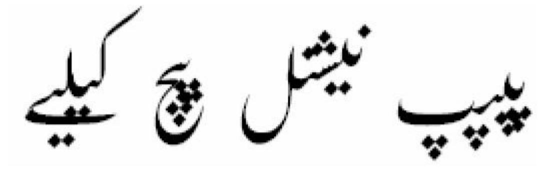


Figure 14: Nuqta placement at default positions

There are 26 characters in Urdu that have nuqtas, as shown below; character *Yeh* (ی, U+064A) has nuqtas at only its initial and medial position.

ب، پ، ت، ٹ، ث، ج، چ، خ، ڈ، ذ، ڑ، 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. Our investigations found that the following characters influenced the nuqta positioning due to the shape of their glyphs.

ے، ج، چ، ح، خ، ف، ق، ع and ک

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

Nuqta clashes are removed according to following observations.

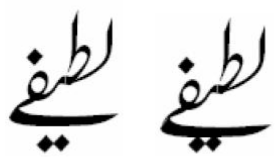

(b) (a)

Figure 15: Nuqta placement for *Yeh Barree*

- The nuqtas of final letters are usually not displaced.
- The nuqtas of isolated letters are usually not displaced.
- The nuqtas of *dad* (U+0639) and *zah* (U+0638) are not displaced.
- Nuqtas of initial letters are preferably placed in their position.
- Nuqtas' clashes with neighboring characters 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 horizontally.

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 the right place without clashing with another nuqta or the boundary of a glyph.

The test results are shown in table 5.

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

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

Acknowledgement

We would like to thank the Nafees Nastaleeq font development team, especially the calligrapher Mr. Jamil-ur-Rehaman who created the beautiful glyphs for this font. The beauty of this font gave us the inspiration to provide 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/workingroups/presentation.htm`

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

Appendix A

Characters in scope are listed in the table below.

U+0622	U+0627	U+0628	U+067E	U+062A
U+0679	U+062B	U+062C	U+0686	U+062D
U+062E	U+062F	U+0688	U+0630	U+0631
U+0691	U+0632	U+0698	U+0633	U+0634
U+0635	U+0636	U+0637	U+0638	U+0639
U+063A	U+0641	U+0642	U+06A9	U+06AF
U+0644	U+0645	U+0646	U+06BA	U+0648
U+06C1	U+06BE	U+0626	U+06CC	U+06D2
U+06F0	U+06F1	U+06F2	U+06F3	U+06F4
U+06F5	U+06F6	U+06F7	U+06F8	U+06F9