

John Hudson, Tiro Typeworks, [john@tiro.ca](mailto:john@tiro.ca)

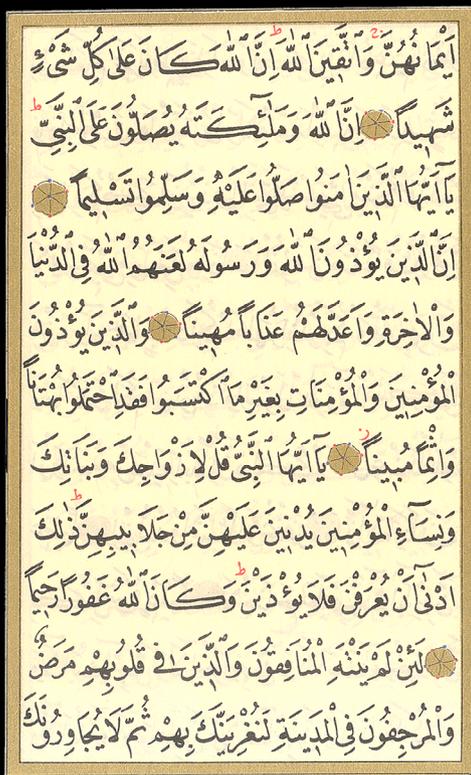
This presentation was originally delivered at TypeCon, Washington DC, Sunday 3 August 2014.

The slides shown in this document were preceded by a short video showing an example of *nasta'liq* calligraphy by the Iraqi-American calligrapher Haaj Wafaa. This video can be viewed on Haaj Wafaa's YouTube channel. [🔗](#)

# PROBLEMS of ADJACENCY

TypeCon 2014

You've just watched somebody writing—a professional scribe, writing with great care—, and there are many things that could be pointed out about that performance. One of them is what I want to talk about today, which is shapes in various kinds of adjacent relationships, the problems that arise in those relationships, and the ways in which those problems are solved. I'm going to be talking about typography, but I've started with this film of someone writing because it's a useful reminder that relationships of adjacency and the problems associated with them are not specific to particular technologies. They are inherent in writing systems. They arise from the variety of shapes employed in a script and the ways in which they combine. It is the solutions to the problems—or the failure to provide solutions—that are specific to this or that technology.



On the left is a manuscript in which a scribe has resolved a great many problems of adjacency—not only of letters but of multiple kinds of marks—, with great skill and confidence. He has learned how to do this, learned what solutions may be used in each situation, so that it is second nature: something he doesn't need to think about each time the problems arise. This allows him to write fluidly and accurately, and in the Islamic *ijāzah* system he will have received certification of his knowledge and skill in this particular style of script.

On the right is the same text, typeset in a technology that also does a good job of resolving these problems of adjacency,\* and which does so using the same canons of competency as the scribe. The difference, of course, is that the knowledge and skill that reside in the mind and hand of the scribe must, in the case of typography, reside to a large degree in the typesetting system. This doesn't make obsolete, of course, the knowledge and skill of the human typographer, the person responsible for the overall text design. But we've come to expect our fonts and our software to take care of a great deal of what we term 'micro-typography' and, at the least, to provide acceptable default behaviour. And micro-typography is all about relationships of adjacency.

\*This is DecoType's ACE technology for Arabic typography. See pages 28–30 for more discussion of this system. Note: if the text in the ACE image looks garbled, this is due to a PDF embedding issue that may be particular to your PDF viewer. Some readers have reported this, but it looks fine on every system I have tested.

# Verification

In this talk, I'm going to be illustrating issues arising from adjacency in structurally complex writing systems from India and the Middle East. But here are some problems of adjacency from our own, relatively simple script, with which I'm sure you're all familiar. The shapes of some letters, when adjacent, need to have their spatial relationships managed. For the scribe, these problems are easily resolved: it is simply a matter of touching the pen down in a specific place, or of manipulating the shapes of adjacent letters to resolve their relationship.

Gerritt Noordzij defined typography as 'writing with prefabricated letters', but I want to suggest that this is something more like an ideal, an image of a typographical technology that is actually able to do what writing does: not in order to be stylistically nostalgic or to mimic handwriting, but to be able to claim the knowledge and skill to implement a writing system as a system.

# V e r i f i c a t i o n

In fact, for most of the past 560 years, typography has been writing with prefabricated rectangles. This is the Gutenberg paradigm: the arrangement of lines of rectangles bearing images of letters and other signs, butted up against each other, creating word images. It is in part testament to the brilliant simplicity of the paradigm that it has persisted for so long. It is also testament to some other things: to the commercial expediency of inheriting designs, spacing and layout models from older technologies; to a tradition of jerry-rigging mechanisms designed for one writing system to approximate the needs of others; and to what begins to look like a lack of imagination for how we might do things differently. If I am now about to start talking about side-bearings, kerning and ligatures, I'd like to pose this question to the future: 560 years from now will we still be talking about typography in these terms?

Verification

This is how these problems of adjacency are typically solved in today's digital fonts. The *V+e* combination is kerned, which is to say that the width of the rectangle of the *V* is reduced, allowing the rectangle of the *e* to intrude into that space. The *f+i* combination is resolved by making it a ligature, which we tend to think of in terms of the graphical linking of the two letter shapes, but which technically is any instance of two or more characters sharing a single rectangle.

These are solutions afforded by the Gutenberg paradigm as translated into successive typesetting technologies. And typographers will tend to look at these kinds of adjacency problems and say 'This needs kerning' or 'This should be a ligature'; that is, to describe the problems in terms of their familiar technological solutions. And it's not wrong for a typographer to think that way if these are the solutions to hand. But it becomes a problem when *technologists* look at the problems in this way, because they will then tend to ask 'How do we implement kerning and ligatures?' instead of asking 'How do we manage relationships of distance and shape?'

సభ్యులవారి

Telugu is a language of south-eastern India. It is one of six Indian languages officially designated as a classical literary language, and is the fifteenth most spoken language on the planet, with approximately 74 million speakers. Like English, it is read from left to right. You don't need to be able to read Telugu, though, to guess that there's something seriously wrong with what we're looking at here. Adjacent shapes are colliding or crowding each other horribly; the word can't be considered legible. Surely some kind of kerning or perhaps ligatures are called for? In fact, what we're looking at here is Telugu characters without any Indic shaping engine or font layout table support.

నాస్తికం

This is what the word should look like, as implemented in the Murty Telugu type that Fiona Ross and I made for Harvard University Press. To explain what's happening here, I'm going to focus on the middle part of the word.

నాక్ష్త్రం

న

0C2B

క్ష

0C4D

ట

0C1F

క్ష

0C4D

వ

0C35

ం

0C47

This is an orthographic cluster, consisting of a conjunct of three consonants plus a vowel, and it's encoded as the sequence of Unicode characters shown below.

నాపపర్

ప

0C2B

్

0C4D

ట

0C1F

్

0C4D

వ

0C35

్

0C47

If I colour-code the three glyphs we use to display this cluster, you can get a better sense of how the glyph display relates to the character string. The first thing you'll likely notice is that the orange element actually represents two characters at either end of the encoding sequence. This is one of the frequent sources of difficulties for complex script display: adjacent on the page or screen doesn't necessarily mean adjacent in the text encoding. The blue and green glyphs each represent a combination of a letter with a preceding vowel killer—a *virama*—, which in this instance serves as a kind of control character triggering specific forms of these letters. These forms indicate to the reader that these consonants are part of the conjunct, to be read as phonetically following the first, orange consonant and before the orange vowel sign, even though the latter ends up positioned slightly to the left of that first consonant. Confusing?

ఫకవ

ఫ

0C2B

క

0C4D

వ

0C1F

క

0C4D

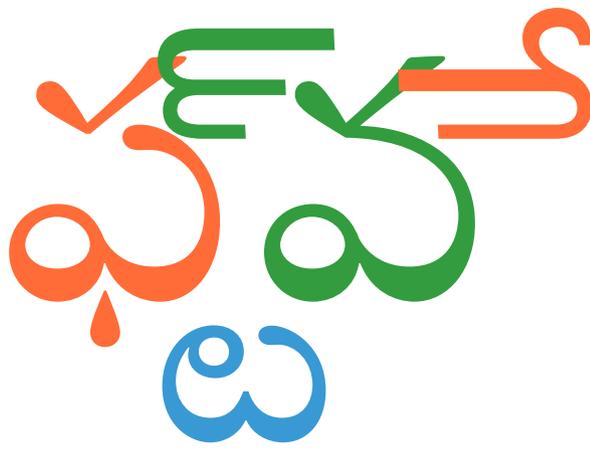
వ

0C35

క

0C47

Let's go through the shaping of this cluster step by step. This, as we saw before, is the character string without any shaping applied. By this stage, we can presume that software will have performed script itemisation on the text and identified this as Telugu, based on the Unicode characters used, and will have passed the text to the appropriate shaping engine. The shaping engine then analyses the character string to identify individual orthographic clusters such as this one, based on a set of rules for how clusters begin and end and what they may contain. This is the stage at which the shaping engine starts to apply OpenType Layout features in the font, running through the glyph string iteratively, looking for matches in glyph substitution lookups.



0C2B



0C4D+0C1F



0C4D



0C35



0C47

The first feature that affects this cluster is the Below Base Forms feature, which matches the blue combination of *virama* plus consonant and substitutes the subscript form. Technically, this is a kind of ligature substitution—one glyph representing two characters—, but visually very different from what we usually think of as a ligature in Latin script.



0C2B



0C4D+0C1F



0C4D+0C35



0C47

The same feature then substitutes the postscript form of the green *virama* plus consonant glyph sequence. You'll notice the presence of the grey dotted circle in the encoding diagram. This indicates that both the blue subscript and the green postscript glyphs are classified as marks. This means, among other things, that they are zero-width glyphs. Hence, they overlap.



0C2B



0C4D+0C1F

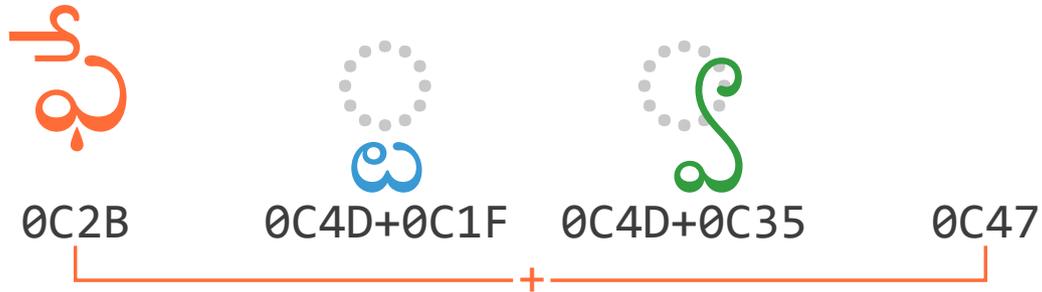
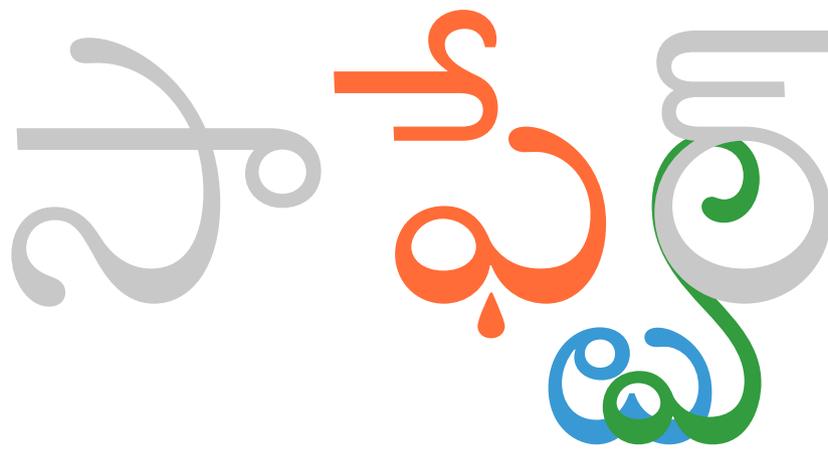


0C4D+0C35

0C47



The next step is to resolve the relationship of the orange initial consonant and final vowel sign. This is done in the Post Base Substitutions feature, and is also a ligature operation. The tricky aspect, though, is that these two glyphs are still at opposite ends of the cluster. They are not adjacent. They can only be ligated by virtue of the fact that the intervening subscript and postscript glyphs are marks, and the OpenType lookup architecture includes a flag setting that allows one to ignore mark glyphs during a substitution or positioning operation. This seems, on the surface, a useful mechanism for this sort of situation, because it allows us to specify the input as simply the base consonant glyph followed by the vowel sign glyph, which is what we have here if we ignore the blue and green mark glyphs. But it is a mechanism that requires any ignorable glyph to be classified as a zero-width mark, even if ultimately that glyph does not behave like a mark, which is clearly the case for the green postscript letter.



This becomes more obvious if we restore the word context in which this cluster occurs. And this brings me to the first observation of how OpenType could have been better than it is at resolving problems of adjacency. It seems to me that it should have been possible to define any arbitrary group of glyphs in a font as being ignorable in a particular lookup. We already have group definitions in OpenType—used for class kerning for example—and we can already define groups of marks to be processed in individual lookups. So could we perhaps extend that mechanism to be able to define arbitrary filter sets, to process or not, regardless of what kind of glyphs are involved?

This slide shows how the cluster looks after all glyph substitution features have been processed. We're left with three glyphs that now need to have their spatial relationship resolved using glyph positioning features.

నా కష్టం

కష్టం

ం

ం

0C2B

0C4D+0C1F

0C4D+0C35

0C47



The first thing we do in positioning is to add an advance width to that green postscript letter; that is, we take the zero-width mark glyph, and we make it a spacing element.

నా క్షయం వర్

క్షయం

యం

ం

0C2B

0C4D+0C1F

0C4D+0C35

0C47



The next thing we do is a kerning operation between the blue subscript and the green postscript. Technically speaking, this is very similar to what we just did to the postscript letter—we're increasing the width of the subscript glyph—but we're defining it in terms of a pair relationship, so it happens only in this particular sequence.

నా కష్టం వార

కష్టం

ట

వ

0C2B

0C4D+0C1F

0C4D+0C35

0C47

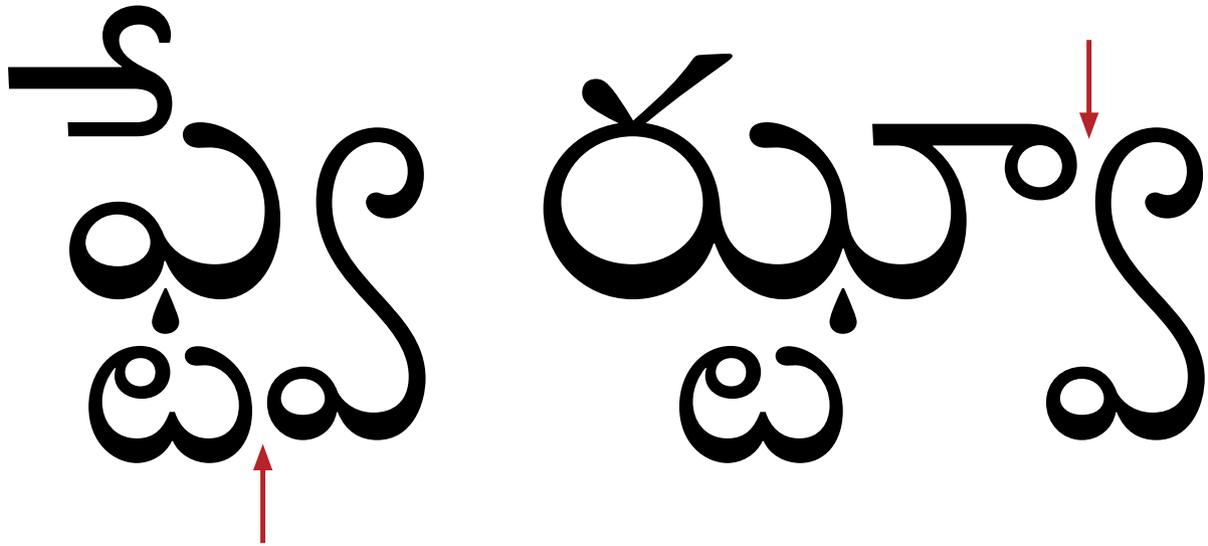


Then we need to position the blue subscript underneath the orange base glyph, which is done with a mark-to-base anchor attachment. And there we have the resolution of the various problems of adjacency within this particular cluster. Now we just need to resolve its relationship to those grey ligatures on either side with a bit of kerning, and we're done...

సాఫ్ట్వేర్

sā - phṭvē - r

In case you're wondering what this says, it's the word *sāphṭvēr*, or 'software'. Which raises an important point to be aware of: very often the borrowing of words from foreign languages will produce combinations of characters that do not occur in the native language of a script, or which are very rare. This means that the writing system as used in today's globalised world may produce problems of adjacency that the system did not evolve to handle during its scribal or earlier typographic development. In the Telugu language, three-consonant conjuncts of this form are relatively uncommon, but in modern newspapers or websites many loanwords from English occur, increasing the need for fonts to be able to handle arbitrary clusters.



Telugu conjunct cluster display is an example of something that is complex and non-trivial to resolve in OpenType Layout. It requires a carefully coordinated sequence of glyph substitution and positioning operations. The situations can get contextually more complex, as I've shown here. In the first cluster, the postscript letter has to kern to the subscript, but in the second, it has to be spaced relative to the base, ignoring the subscript. The subscript and postscript are the same in each case, but their relationship is determined by the base to which they are adjacent. This has to be handled using context strings in the kerning lookups.

The overall process of displaying Telugu clusters with OpenType involves a couple of non-intuitive and apparently redundant steps, but it is all reasonably do-able. OpenType might not provide the best mechanisms to resolve the problems of adjacency in Telugu, but they are sufficient.

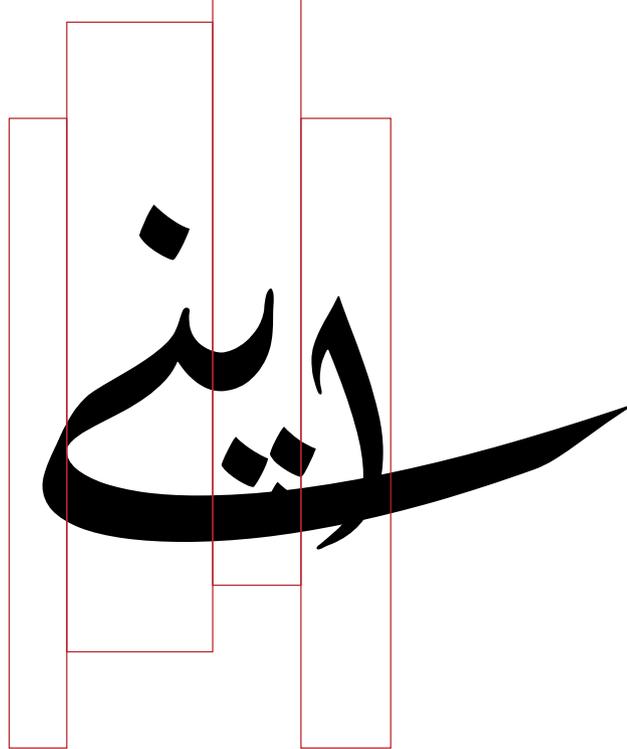


This is Aldhabi, a display typeface in an Ottoman *diwani* style designed by the brilliant Tim Holloway, with input from Fiona Ross and Mamoun Sakkal, which we valiantly tried to make into a font for Microsoft a couple of years ago. Or, rather, this is what Aldhabi is supposed to look like. I've had to engage in some significant manual adjustments in order to get it to look this good, in order to resolve problems of adjacency that OpenType can't reasonably handle.

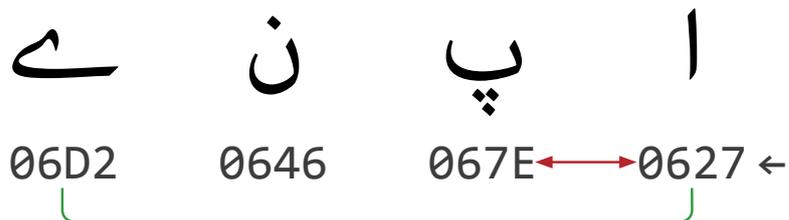
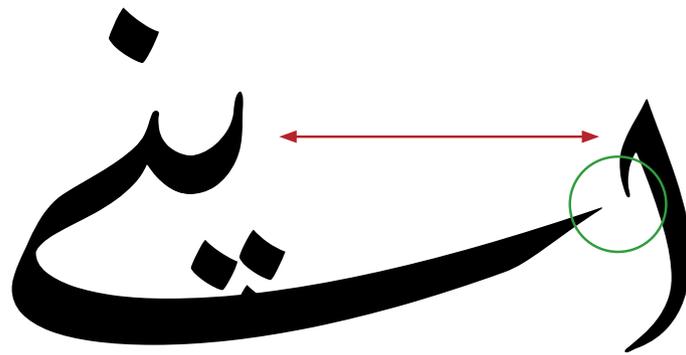
A version of Aldhabi does ship with Windows 8, and quite a lot of things do work. It can be used, with some care and occasional fiddling, for short pieces of Arabic and Persian text such as headlines and banners. But don't try using it for Urdu, the national language of Pakistan.



This is how this Urdu word—the common third person plural pronoun—displays with the currently shipping version of Aldhabi. Again, one doesn't need to be able to read the language in order to see that something is seriously wrong here. Visually, the problems are those created by the last letter in the word: the long stroke that swings back to the right. This is the *barī yē*, literally the 'turned *yē*'. This is a form that originated as a stylistic variant of the Arabic letter *yē*, but which in the Urdu alphabet became a distinct letter, representing a long *ē* vowel sound. The shape of this letter results in numerous problems of adjacency, but the intense difficulty of resolving these problems in OpenType results from technical constraints and, I would argue, an inappropriate paradigm...

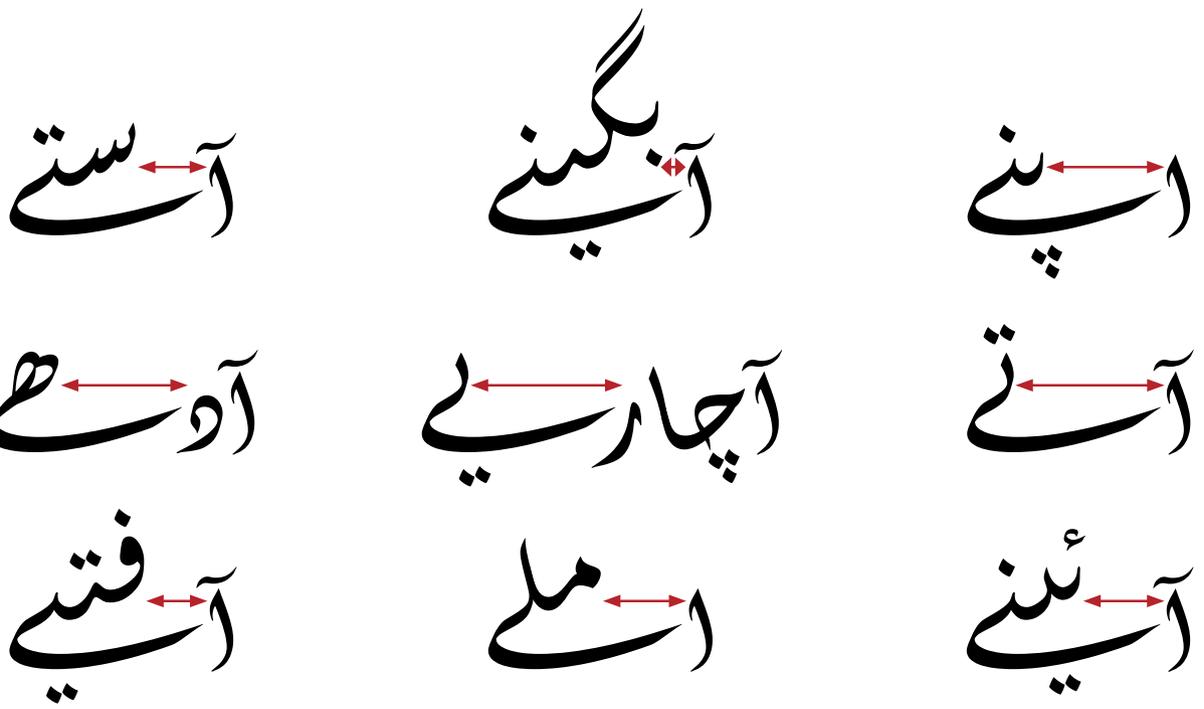


It's those prefabricated rectangles again. Using OpenType cursive attachment positioning, we're able to vary the heights of the rectangles to quite elegantly handle the ways in which letters connect, but the process of what software makers call 'line layout' is still essentially one of butting up rectangles against each other, and the only available mechanism to fix problems of adjacency arising from this process is still, basically, kerning.



The difficulty with kerning in this situation is that the desirable relationship of visually adjacent shapes, indicated by the green circle, is only achievable by applying a positive kerning adjustment between adjacent glyphs, indicated by the red line. To put it another way, in order to achieve a particular relationship between the first and fourth letters, you have to kern the first and second letters.

This is theoretically do-able with OpenType glyph positioning. It is possible to define a kerning pair between the first and second glyphs that is contextually dependent on the following glyphs. Practically, though, it is closer to impossible, because you would need a separate contextual lookup for every specific distance adjustment required to achieve that visual relationship, each with its own set of one or more context statements involving anywhere from one to four following glyphs.



Here is a small set of Urdu words in which the *baī yē* occurs. As you can see, the amount of kern adjustment required to solve the problem of adjacency in each word differs, affected by the quantity and widths of the context glyphs and by the identity of the glyphs to be kerned. When working on the Aldhabi font, I fairly quickly realised that this isn't something a human being can do. The amount of work involved to figure out the distances and specify all the kerning lookups with their contextual statements is immense. So the obvious thought is that it might be something that could be automated, something that a computer could do based on the available spacing and cursive attachment data. Maybe. My guess is that you'd overrun the GPOS table fairly quickly, and the font wouldn't compile without some significant fiddling with subtable boundaries. The impact on line layout performance speeds might also be significant, as the layout engine runs through hundreds of GPOS lookups looking for contextual matches. But if somebody wants to give it a try, do let me know.

اپنے

ے

06D2

ن

0646

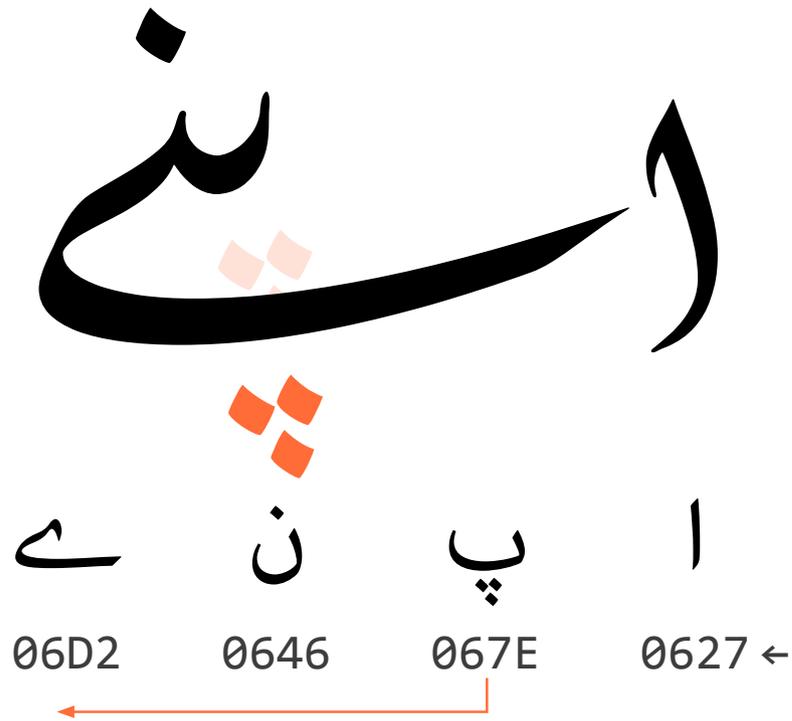
پ

067E

ا

0627 ←

The problems of adjacency caused by the *barī yē* are not limited to spacing. Dots and other marks below the letters preceding the *barī yē* will be obscured or crowded by the turned stroke.

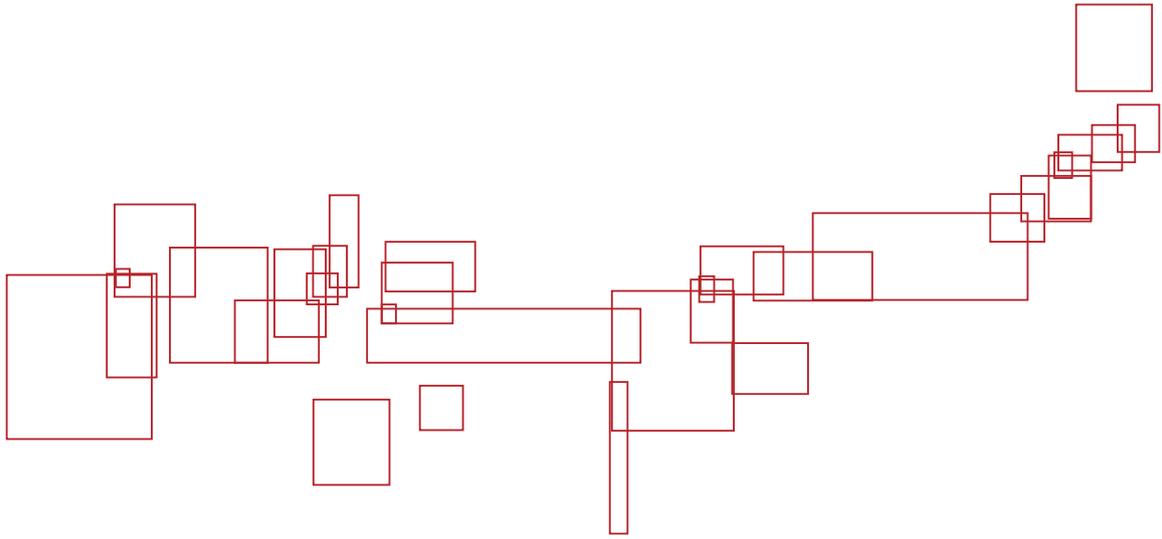


The convention is to lower them, so that they sit below the *ba'ir yē*, while still aligned with their base letter shape above. As you can probably imagine, this is a complex contextual operation in OpenType terms. First, the dots have to be decomposed from their base letter and handled as marks. Then they have to be dropped a precise distance determined not only by the presence of the *ba'ir yē* but also any intervening letters. As with the spacing problem, the quantity and complexity of the lookups involved is huge, presenting challenges for workflows, font compilation, and performance.

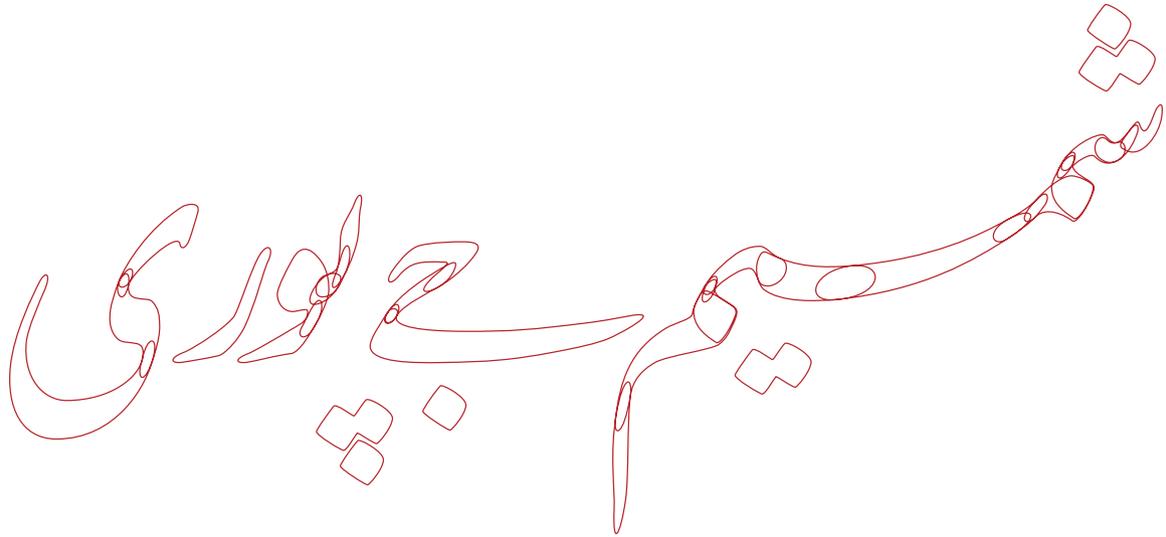
Last year, Kamal Mansour at Monotype explained to me the solution he had come up with for this problem, which involves contextually removing the dots from their base and reordering them after the *ba'ir yē* in a way that tracks to which preceding letter they belong. The marks are then horizontally positioned using a limited set of anchor positions under the *ba'ir yē* stroke. It is a clever workaround that results in a reasonable approximation of the ideal result (an approximation because the horizontal anchors are no longer precisely aligned with the base letter above). But it is still a workaround: a convoluted hack to bypass the limitations of an inappropriate paradigm.

تسمہ ہے پوری

This is the title of an Urdu poem set in a Nasta'liq typeface designed by Mirjam Somers. It is not merely a change of font, though, but a change to an entirely different typesetting system and, indeed, a different paradigm. This is DecoType's Arabic Composition Engine, as implemented in the Tasmeeem plug-in for InDesign.



If we were to visualise how DecoType's system works in terms of rectangles, for comparison with the Gutenberg paradigm, it would look something like this.



But the genius of the DecoType system is that there are no rectangles. There are only the shapes from which the letters and words are composed, arranged in specified relationships, with spacing of adjacent shapes handled not via sidebearings, kerning adjustments or contextual anchor positions, but via flexible distances and tolerances. It's a technology made by people who know a lot about two things: the Perso-Arabic writing system, and computer programming. And when it comes to problems of adjacency—and much else—it runs rings around OpenType. I present it as an example of what can be achieved by people who don't know anything about what most of us think of as typography, and so are not constrained by assumptions or by inherited paradigms. When I look at this image, I am aware that I am looking at a natively digital typesetting technology: something new, and un beholden to yesterday's machinery and fonts.

# ঈর্ষাৎ

Bengali, used by more than 300 million people in north-eastern India, is a good example of a writing system with inherent problems of adjacency. It is, in the words of the newspaper publisher Aveek Sarkar, 'a house with too many tenants in the upper storey'.

This isn't a real word, but it illustrates three kinds of above-line collisions that occur with some frequency. In a surprising amount of Bengali typesetting, this kind of collision is simply accepted.

স্বীর্থাটদি

স্বীর্থাটদি

Here is how we resolved these problems in a headline typeface that we made for Mr Sarkar's newspapers. The font contains a variety of variant forms of letters with trimmed, shortened, lowered or raised above-line features that provide tidier, more legible combinations. We also include contextual final forms of some letters, so that when they occur at the end of a word they drop the connecting head line stroke on the right, for a cleaner shape.

This is the sort of thing that OpenType Layout is actually very good at. The substitutions are simple one-to-one mappings, and the contexts are seldom more than one glyph deep; marks can be ignored or not as appropriate to the particular substitution.

স্বীর্থাট্টি

স্বীর্থাট্টি

When we came to test the typeface in InDesign CS6, however, we discovered that only some of our contextual substitutions were being applied, while others were being triggered where we didn't want them. And as I sat and looked at this I realised that it was evidence of something really, really unfortunate.

The contextual substitutions that were working correctly were all happening inside individual syllabic clusters, while every time the context involved looking across a cluster boundary the substitutions were failing, or were being triggered after every cluster if they involved exception statements. This meant that many of our contextual shortened flourish forms were not appearing where needed, while our word-final forms were appearing even in the middle of words.

স্বী×থ×ট×ছি

That's when I realised that Adobe's World Ready Composer shaping engine was applying all Indic layout features at the discrete cluster level, as if each cluster existed in isolation, which meant that no glyph substitution lookups involving cross-cluster contexts would ever be triggered, while the end of every cluster would be treated as if it were the end of a word for final-form substitutions. Then things got worse, because it turned out that not only Adobe's shaping engine behaved this way, but so did Microsoft's, Apple's, and the open source Harfbuzz engine. The OpenType Indic shaping specification, which requires some specific features to be applied at the cluster level in order to correctly shape syllables, had been interpreted by software makers as meaning that all layout features should be applied at this level and only at this level.

The Sarkar Bengali type is mostly used for newspaper headlines. So we worked around this bug by disabling the contextual final forms, and implementing these and the shortened flourishes as discretionary stylistic set features. This means that the typesetter needs to selectively apply these features, rather than being able to rely on the font and layout engine to provide appropriate micro-typographic defaults.

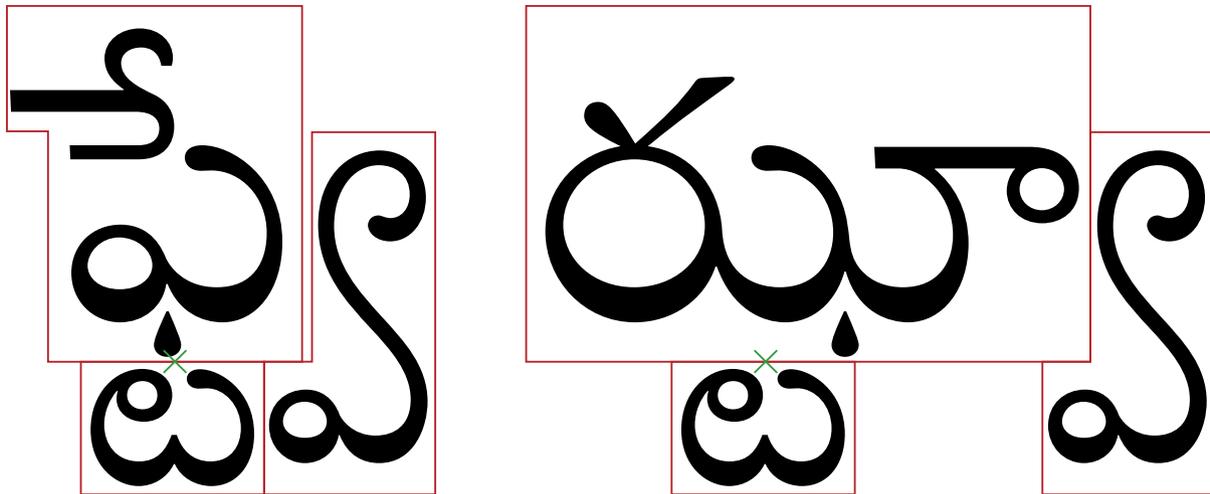
ਉਪਿੰਰਉਯੇ  
ਉਪਿੰਰਉਯੇ

This isn't a workaround that would be practical for the Murty Gurmukhi typeface for Harvard University Press, which will be used for large amounts of text. It would be unreasonable to expect the typesetter to manually apply selective substitution lookups. So, in order to affect the cross-cluster adjustments needed to avoid collisions and crowding of the kinds illustrated here—again, not a real word, but representative of real problems—, Fiona and I carefully defined a set of six stylistic set features to insert head line extenders of different widths in specific situations. Then Karsten Luecke wrote a script that runs in InDesign and seeks the specific sequences of Unicode characters in the text to which each of these stylistic sets needs to be applied. It's one of the weirder hacks I've ever had to come up with.

To their credit, all of the shaping engine makers have acknowledged that this was an error on their part, and that it needs to be fixed in order to facilitate Indic typography. At a meeting in Seattle in April, we worked together to define a way to do this, and I'm hoping within the next months we'll have some testing capabilities set up and will be able to provide examples of how fonts need to be made in order to take advantage of the fix. I can't report when this will be fixed in shipping software though, and in the meantime the bug stands as a reminder that even when the OpenType Layout format makes something easy to achieve in a font, there's no guarantee that it is going to work in the wild.

పీర్స్ రుక్మా  
టివి

I'm going to conclude with some thoughts on ways in which some problems of adjacency might be better handled than in OpenType as currently defined. I've already suggested that the existing OpenType Layout architecture could be improved by allowing for filtering of arbitrary glyph groups in lookup processing. And I've shown Decotype's ACE system as an example of an entirely different paradigm, built around analysis of the behaviour of a specific writing system. What follows is by way of fanciful speculation: I don't offer it as a fully thought-out model for how things should be done; rather, as a suggestion of how they might be done. And I wanted to reassure you, in case you wondered, that I don't have anything fundamental against rectangles.



You can go quite a long way with rectangles, especially if you start cutting bits out of them. You can, in fact, handle fairly subtle relationships of adjacency by butting such shapes up against each other, especially if done so in combination with anchor attachments, as shown here. If we move away from the idea that some glyphs are spacing and some are zero-width, and instead think in terms of glyphs having envelopes, we might bypass many of the contextual positioning bottlenecks that plague OpenType font development for complex scripts. Nor do the envelopes need to be based on rectangles: they could be balloons, or any kind of polygonal shape that results in the desired default relationships between adjacent shapes. We've already employed something like this with so-called 'cut-in kerning' in mathematical typesetting fonts for Microsoft. But rather than being a specialised adjustment method applied to the existing sidebearing model, it could be the underlying spacing mechanism, replacing the whole concept of sidebearings and better accommodating the great variety of shapes that the world's writing systems present to our ingenuity.

And this seems a good place to stop. It was twenty years ago that Ross Mills and I formed Tiro Typeworks: twenty years in which I have been privileged to make my living designing typefaces and making fonts. As I created this image, I realised that it was also almost twenty years ago that Laurence Penney and I first chatted about the idea of glyph envelopes, on the comp.fonts Usenet newsgroup. I don't know if it is an idea whose time has come. Perhaps it is just one possible idea to contribute to OpenType 2.0? You may have your own.

# Thank you

ధన్యవాదాలు

آپ کا شکریہ

তোমাকে ধন্যবাদ

बुगुडा यंनहाद

For more examples of Iraqi/US calligrapher  
Haaj Wafaa's work

[www.youtube.com/user/koofah](http://www.youtube.com/user/koofah)

[www.maskindesign.com](http://www.maskindesign.com)

For more information about DecoType's  
Arabic Composition Engine

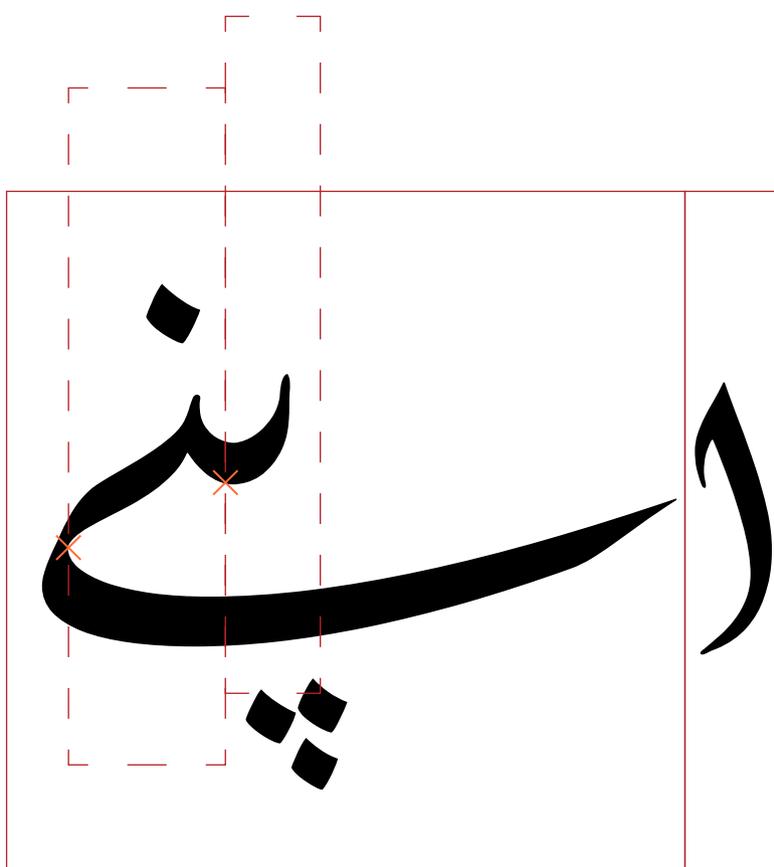
[www.decotype.com](http://www.decotype.com)

For a rag'n'bone assortment of documents  
and images, some of which relate to the topics  
of this presentation

[www.tiro.com/John](http://www.tiro.com/John)

Thank you.

[Read on for the Afterword. →](#)



### Afterword

In the Q&A session following my presentation at TypeCon, Nadine Chahine asked if the existing, rectangle-based spacing model could be adapted to situations such as that of the Urdu *barī yē*, rather than replaced with some different model. This is something I have also pondered, and the above illustration shows how this might be done. The cursive attachment connections (orange × signs) enable identification of a kind of ‘type object’ from the connected glyphs, and rather than spacing of preceding letters being based on the sidebearing of the first glyph in the object it is determined by the right-most sidebearing of the object (in this case the large rectangle of the *barī yē*). In this situation, the rectangles of the first two letters in the connected object have no effect on spacing. [The dot repositioning remains a challenge, and is not discussed here.]

I think there is some merit in this idea of allowing certain kinds of glyph connections—cursive attachment and anchor attachments—to define type objects at a level higher than that of individual glyphs. These objects could then be addressed in a variety of ways during line layout. One can start to conceive of object-to-object kerning, for example, or even hybrid object-to-glyph kerning, and of object classes that could be defined in terms of potential rather than explicit glyph composition.