Public Review Issue #30

Encoding of Bengali Khanda Ta in Unicode

Peter Constable, Microsoft Corporation
2004-02-17

This document considers requirements for encoded representation of the Bengali khanda ta and some alternate encoding models, and presents one of these as a proposed solution. The need for this has arisen primarily due to confusion surrounding the current description of khanda ta in section 9.2 of Unicode 4.0; in particular, due to references to khanda ta as a half form.

Four alternate models are considered:

A. Khanda ta represented as < ta, virama > with ZWJ for explicit override

B. Khanda ta represented as < ta, virama > with ZWJ, ZWJ + ZWNJ overrides

C. Khanda ta represented as < ta, virama, ZWJ >

D. Khanda ta encoded as a separate character

Model A is the model that is currently intended by the specification in section 9.2 of Unicode 4.0 for representing khanda ta. It is proposed that Model A be kept, and that the text be revised to clarify the intended behaviours and to remove any ambiguities.

Background

Khanda ta is a letterform used in Bengali for a consonant ta with killed vowel. It has the same phonological value as ta-hasanta, though usage conventions for these differ.

Early in the 20th century, it was reported by Chatterji (1926) that ta-hasanta was preferred for indigenous Bengali words (i.e. those derived from Prakrit) in contexts in which conjunct forms would occur for loans from Sanskrit, Persian or other languages. Khanda ta originated, apparently, as an alternate way to write ta-hasanta in such contexts. (It is not clear to me at this time when this innovation occurred.)

In modern Bengali-language usage, khanda ta would be used for words of Prakrit or Sanskrit origin, but it is reported that some would not use khanda ta for Persian, English or other loans, preferring ta with overt hasanta. Some reportedly also use ta with overt hasanta for onomatopoeic words. Both forms are also reported to be used in some other languages written with Bengali script. For instance, khanda ta is used in Sylheti-language text in Bengali loan words or in cognates that retain the same form in both languages, but otherwise ta-hasanta is reportedly used. The need for both forms also exists for use in pedagogical texts.

As it is a later innovation in the development of Bengali script, khanda ta is not used in older texts, and would not normally be expected in Sanskrit-language documents.

Khanda ta should not be considered a "half form" since the notion of "half form" as found in Devanagari does not apply in the same way to Bengali script, and it can be misleading, suggesting the formation of a cluster with the following consonant. Use of terminology is not always consistent, though, and some may refer to it this way. Apart from its visual appearance, its writing-system behaviour is like ta-hasanta: following the conventional behaviours of Bengali script, it does not take matras, and matras do not re-order around it. It does provide a base for reph; thus, a cluster-initial reph is written on the khanda ta and not on a following consonant.

The following are the known behaviour requirements related to khanda ta:

- Ta + virama normally ligates with following ta, tha, na, ba, ma, ra and la.
- A half-form reduced-size ta glyph may have existed in some font implementations for creation of certain connected-conjunct forms,[1] but a non-connected reduced-size ta is not known to be in attested usage. It is a *non*-requirement, however, to prohibit display of such a form.
- Both ta-overt hasanta and khanda ta are attested; they need to be distinguished in encoded representation as they can occur in comparable contexts in a given document. This distinction cannot be attributed to cultural conventions or font variants.
- Where either ta-overt hasanta or khanda ta occur, an orthographic syllable (i.e. cluster) boundary follows. Thus, in a sequence such as < ta, virama, ssa, e-kaar > the matra appears to the left of the ssa, not the killed ta. A preceding < ra, virama > results in reph positioned on the killed ta.

One implication of these requirements is the need to encode a three-way distinction in the context of a following consonant such as na with which ta form a ligated conjunct form.:

ত্ন ৎন ত্‌ন

As stated above, however, it is assumed that a fourth distinction for a reduced-sized "half" ta is not needed.

Alternate models for encoding khanda ta

The following alternate encoding models are considered:

A. Khanda ta represented as < ta, virama > with ZWJ for explicit override (the existing Unicode specification)

B. Khanda ta represented as < ta, virama > with ZWJ, ZWJ + ZWNJ overrides

C. Khanda ta represented as < ta, virama, ZWJ >

D. Khanda ta encoded as a separate character

[1] In this document, the terms "conjoin" and "conjunct" are used to imply a graphically-connected (ligated) letter form, not merely a written representation of a phonological consonant cluster.

Other possibilities exist (e.g. representation as < ta, ZWJ, virama >), but are either known to fall short of the requirements above or to involve innovations into the general Indic encoding model that are not necessary, and so are not considered here.

In order to give a complete account of the interpretation of encoded character sequences with ta, it is necessary to consider a range of contexts in which ta and its related forms can occur. In particular, orthographic syllable (cluster-boundary) behaviours and rendering in different encoded sequences must be considered in these contexts:

- before a consonant with which ta forms a conjunct
- before a consonant with which ta does *not* forma a conjunct
- before ZWJ or ZWNJ
- at the end of a word

In the description below, the following notation will be used:

Notation	Meaning
Cj	consonants that conjoin with ta
Cn	consonants that do not conjoin with ta
C	any consonant character
#	end of word
¦	cluster (aksara) boundary (determining the positioning of reph and matras)
*	any string of characters (or resulting glyphs)

In order to illustrate clearly the behaviour in relation to orthographic-syllable boundaries, some example sequences that include e-kaar will be used. For the class Cj, ta ত is used as a representative character. For the class Cn, ka ক is used as a representative character.

In the descriptions of each model, certain displayed results will be described as “fallback rendering”. The rendering shown is not what would normally be expected, though a rendering system would be permitted to display in that way. For instance, a rendering system might use an overt-hasanta form if a particular conjunct ligature is not supported. Rendering systems would not be *required* to support any kind of fallback rendering, however.

In evaluating advantages or disadvantages of each model, one recurring consideration is the implications a model has for implementations in relation to processes that are sensitive to cluster boundaries. The issues involved in this are described in greater detail in an appendix.

Model A: khanda ta as < ta, virama > with ZWJ for explicit override

In this model, khanda ta is typically encoded simply as < ta, virama >. In the contexts in which clear differentiation is needed, i.e., before conjoinable consonants such as na, ZWJ is used to override conjunct formation. This is the model that is currently specified in Unicode 4.0 *as it is intended to be interpreted.*

Full details regarding interpretation of different sequences involving ta under this model are summarized in the following table.

Pattern	Sample text sequence	Display	Note
< ta, virama, Cj, ekaar >	< ত, ◌্, ত, ে◌ >	¦ত্তে¦	
		ৎ¦তে	fallback rendering
		ত্¦তে	fallback rendering
< ta, virama, Cn, ekaar >	< ত, ◌্, ক, ে◌ >	ৎ¦কে	
		ত্¦কে	Sanskrit-language text
< ta, virama, # >	< ত, ◌্, # >	ৎ¦#	
		ত্¦#	Sanskrit-language text
< ta, virama, ZWNJ, * >	< ত, ◌্, ZWNJ, ∗ >	ত্¦∗	
< ta, virama, ZWJ, * >	< ত, ◌্, ZWJ, ∗ >	ৎ¦∗	

For junk sequences involving < ta, virama >, < ta, virama, ZWJ > or < ta, virama, ZWNJ > immediately followed by a vowel matra or a modifier, it could be recommended that rendering implementations treat the dead ta as a base for the following mark.

Note that this model supports the three-way distinction required before consonants such as ta with which ta can form a ligated conjunct form:

Sample text sequence	Display
< ত, ◌্, ত, ে◌ >	¦ত্তে¦
< ত, ◌্, ZWJ,ত, ে◌ >	ৎ¦তে
< ত, ◌্, ZWNJ,ত, ে◌ >	ত্¦তে

It is to be noted that the use of ZWJ in this case does not imply the creation of a "half form", or that the khanda ta is to be considered part of the same cluster as the following consonant. This is currently not clear in the description of khanda ta in section 9.2 and in FAQ#19. The proposal using this model, then, would be to improve the documentation related to khanda ta.

Pros. This model has these advantages:

- It adheres to the current specification in Unicode 4.0 and FAQ#19, as these are intended to be interpreted. The problem in the current text relates to a *possible* perception that < ta, virama, ZWJ > should form a cluster with the following consonant, but anyone familiar with the script knows that is not an expected or required behaviour.
- It supports all necessary distinctions and behaviours without requiring new characters or *major* departures from encoding of Indic scripts. (It involves an exception to prototypical behaviour of ZWJ, but such exceptions already exist.)

Cons. The disadvantages of this model are:

- It involves an exception to prototypical behaviour related to ZWJ in the general Indic model in that ZWJ is used to represent a letterform that has a following obligatory cluster break. Other exceptions already exist, however.
- Unless a set of known consonants can be fixed in advance, clustering behaviour in < ta, virama, Cj > contexts cannot be predicted from the string, but must depend on information within the font. A set of consonants cannot be fixed, however, since the need for new conjunct forms may arise in the future, and because some conjuncts may be supported in some fonts but not others. This can create particular difficulties for some implementations that perform certain operations, such as determining allowable caret positions, without using full font/glyph-layout processing.

Model B: khanda ta as < ta, virama > with ZWJ, ZWJ + ZWNJ overrides

This model is largely the same as the previous model but differs in an important respect: sequences with ZWJ never involve a cluster break; this leads to an additional sequence for representing khanda ta: < ta, virama, ZWJ, ZWNJ >.

The essential points of this model are take from a proposal submitted to UTC by Gautam Sengupta; cf. document L2/04-060.

Details regarding sequences involving consonants Cj are presented in the following table. Interpretation in all other contexts is identical to that in model A.

Pattern	Sample text sequence	Display
< ta, virama, ZWJ, # >	< ত, ◌্, ZWJ, # >	ৎ¦#
< ta, virama, ZWJ, C >	< ত, ◌্, ZWJ, ত, ◌ে >	¦ৎতে¦
< ta, virama, ZWJ, ZWNJ, C >	< ত, ◌্, ZWJ, ZWNJ, ত, ◌ে >	ৎ¦তে

Like model A, this model supports the three-way distinction required before consonants such as ta with which ta can form a ligated conjunct form. Indeed, it supports a fourth distinction: clustering or non-clustering behaviour (the former not being a requirement):

Sample text sequence	Display
< ত, ◌্, ত, ◌ে >	¦ত্তে¦
< ত, ◌্, ZWJ,ত, ◌ে >	¦ৎতে¦
< ত, ◌্, ZWJ, ZWNJ, ত, ◌ে >	ৎ¦তে
< ত, ◌্, ZWNJ,ত, ◌ে >	ত্¦তে

Pros. This model has these advantages:

- It supports all necessary distinctions and behaviours without requiring new characters.
- It avoids creating an exception in the prototypical behaviour of ZWJ.

Cons. The disadvantages of this model are:

- It deviates from the current specification in Unicode 4.0 and FAQ#19 as these are intended to be interpreted, though it could be claimed that it follows a *different* interpretation of the specification.
- Unless a set of known consonants can be fixed in advance, clustering behaviour in < ta, virama, Cj > contexts cannot be predicted from the string, but must depend on information within the font. A set of consonants cannot be fixed, however, since the need for new conjunct forms may arise in the future, and because some conjuncts may be supported in some fonts but not others. This can create particular difficulties for some implementations that perform certain operations, such as determining allowable caret positions, without using full font/glyph-layout processing.
- It introduces the cumbersome sequence < ZWJ, ZWNJ > solely to preserve prototypical cluster-related behaviour of ZWJ. This would only really be needed preceding Cj consonants, which appears not to be common. Where it *is* needed, though, users would need to learn new entry and editing behaviour.

Model C: khanda ta as < ta, virama, ZWJ >

This model has some similarities with the model A but differs in an important respect: khanda ta is *only* representable using the sequence < ta, virama, ZWJ >.

A question arises in this situation: what is displayed in contexts that do *not* have ZWJ but that would result in khanda ta under model A? The answer generally adopted here is that a "half" form of ta would appear in these contexts. This would be a reduced-size form of the full-ta shape that becomes part of the cluster of the following consonant—the prototypical Indic half-form behaviour for conjunct contexts in which a true ligature-conjunct form does not exist. This reduced-size ta is not known to be required in any usage scenario, but it is not considered a problem to make representation of this text element possible, and it would provide reasonably familiar and predictable behaviour. (Of course, a variation of this model could be devised using a different answer to this question, though such differences are not important for purposes of evaluating alternate models.)

Full details regarding interpretation of different sequences involving ta under this model are summarized in the following table.

Pattern	Sample text sequence	Display	Note
< ta, virama, Cj, ekaar >	< ত, ◌্, ত, ◌ে >	¦ত্তে¦	
		¦ত্তে¦	fallback rendering
		ত্¦তে	fallback rendering
< ta, virama, Cn, ekaar >	< ত, ◌্, ক, ◌ে >	¦ত্কে¦	
		ত্¦কে	fallback rendering
< ta, virama, # >	< ত, ◌্, # >	ত্¦#	
		ত¦#	(reduced-size ta) other possible rendering
< ta, virama, ZWNJ, * >	< ত, ◌্, ZWNJ, * >	ত্¦*	
< ta, virama, ZWJ, * >	< ত, ◌্, ZWJ, * >	ৎ¦*	

For junk sequences involving < ta, virama >, < ta, virama, ZWJ > or < ta, virama, ZWNJ > immediately followed by a vowel (matra) or a modifier, it could be recommended that rendering implementations treat the dead ta as a base for the following mark.

Note that this model supports the three-way distinction required before consonants such as ta with which ta can form a ligated conjunct form:

Sample text sequence	Display
< ত, ◌্, ত, ◌ে >	¦ত্তে¦
< ত, ◌্, ZWJ,ত, ◌ে >	ৎ¦তে
< ত, ◌্, ZWNJ,ত, ◌ে >	ত্¦তে

Indeed, the representation of all three forms in this context is exactly the same as for model A. These two models differ only in *other* contexts.

As in the case of model A, it is to be noted that the use of ZWJ here does not imply the creation of a "half form", or that the khanda ta is to be considered part of the same cluster as the following consonant. This must be made clear in the description for Bengali script in section 9.2 of the Standard.

Pros. This model has these advantages:

- It supports all necessary distinctions and behaviours without requiring new characters or *major* departures from encoding of Indic scripts. (It involves an exception to prototypical behaviour of ZWJ, but such exceptions already exist.)

- Implementations can determine cluster-related behaviour of sequences involving ta completely from the character sequence alone, allowing implementations to perform all related operations without prior knowledge of font-dependent glyph processing.

Cons. The disadvantages of this model are:

- It deviates from the current specification in Unicode 4.0, though does not entirely abandon it.
- It artificially introduces a letterform (the reduced-size “half” form) that is not actually in known usage.
- It involves an exception to prototypical behaviour related to ZWJ in the general Indic model in that ZWJ is used to represent a letterform that has a following obligatory cluster break. Other exceptions already exist, however.
- It varies from prototypical behaviour related to ZWJ in the general Indic model in that there is a cluster-preserving half-form which cannot be rendered using ZWJ; this letterform is not actually in known usage, however.
- The khanda ta would always require a sequence with ZWJ, including word-final contexts and before non-conjoinable consonants. Using existing keyboard input technologies, users would always have to be aware of the role of the invisible control character ZWJ in representation of khanda ta and consciously enter or delete that character; this could only be avoided using context-aware intelligent input methods, technologies that are not widely used except for East Asian text. As a result, users would have to learn new editing practices, which would not likely be well received.

Model D: khanda ta encoded as a separate character

This model clearly differs from the previous two in that khanda ta is encoded always and only as a distinct, atomic character.

The question that arises in this case is what interpretation will be given to sequences that led to khanda ta in previous models. As for model C, the answer adopted here is a reduced-size “half” form of ta. The representation of conjunct forms, overt-hasanta forms and the artificially-innovated “half” form would follow the prototypical Indic model precisely.

A further question arises as to how the khanda ta character would behave in sequences that would be acceptable for typical consonants: followed by matras, candrabindu, ZWNJ, etc. (These would, of course, be considered “garbage” text.) The arbitrarily-chosen answer adopted here is that no changes in clustering, ordering, or glyph shape would result, but that combining marks can be applied. Obviously, there are variations of this model that answer this question differently.

Full details regarding interpretation of different sequences involving ta under this model are summarized in the following table. In this case, it is not necessary to illustrate how khanda ta is encoded; it is only necessary to show how sequences involving ta are to be interpreted.

Pattern	Sample text sequence	Display	Note
< ta, virama, Cj, ekaar >	< ত, ◌্, ত, ে◌ >	¦ত্তে¦	
		¦ত্তে¦	fallback rendering
		ত্¦তে	fallback rendering
< ta, virama, Cn, ekaar >	< ত, ◌্, ক, ে◌ >	¦ত্কে¦	
		ত্¦কে	fallback rendering
< ta, virama, # >	< ত, ◌্, # >	ত্¦#	
< ta, virama, ZWNJ, * >	< ত, ◌্, ZWNJ, * >	ত্¦*	
< ta, virama, ZWJ, C >	< ত, ◌্, ZWJ, ত, ে◌ >	¦ত্তে¦	(reduced-size ta)
< ta, virama, ZWJ, # >	< ত, ◌্, ZWJ, # >	ত¦#	(reduced-size ta)

For junk sequences involving < ta, virama >, < ta, virama, ZWJ >, < ta, virama, ZWNJ > or khanda ta immediately followed by a vowel (matra) or a modifier, it could be recommended that rendering implementations treat the dead ta or khanda ta as a base for the following mark.

Of course, this model supports the three-way distinction required before consonants such as ta with which ta can form a ligated conjunct form.

Pros. This model has these advantages:

- It avoids creating an exception in the prototypical behaviour of ZWJ.
- Implementations can determine cluster-related behaviour of sequences involving ta completely from the character sequence alone, allowing implementations to perform all related operations without prior knowledge of font-dependent glyph processing.

Cons. The disadvantages of this model are:

- It abandons the current specification in Unicode 4.0 in relation to khanda ta.
- It introduces a new character to support a distinction that *can* be represented using existing mechanisms (albeit with some costs); this may lead to expectations that user communities can request characters for reasons related to cultural perceptions of a script rather than any technical requirements.
- It results in khanda ta being represented as an entirely different spelling from other forms of ta. This breaks the phonological and graphological connection between various forms of ta, with some negative impact for searching and for presentation of text in work involving historical texts; e.g., switching presentation between khanda ta and other presentations cannot be done by formatting but requires conversion of the data itself. (From a cultural perspective, this also breaks the historical connection between khanda ta and other forms of ta.)

Comparison of models and proposed solution

All four models are equal in their ability to support representation of required text elements and necessary distinctions between text elements. Their relative merits differ significantly otherwise, however.

While there appears to be particular preference for model D among some members of the Bengali community, it has not been shown that a new character is, in fact, required: there is no text that needs to be represented that cannot be represented without adding the new character. The model has some significant disadvantages, in particular the loss of connection between khanda ta and other forms of ta. Also, the technical advantages are not strong advantages. Unless additional technical advantages can be identified, there is not adequate justification to select model D as the preferred recommendation.

Model C is simpler for implementers to deal with, but at the expense of end users, who would almost certainly have to learn new practices for entering and editing text. As simpler implementation is the primary argument in favour of this model, it would not be appropriate to recommend this model in view of the negative impact on usability.

We are left with model A, the existing specification in Unicode 4.0, and model B. Both may create additional difficulty for some implementations, yet they maintain reasonably-familiar and predictable user experience in most contexts—though model B is less so in the not-very-common case of khanda ta followed by a conjoinable consonant where a cluster break is needed.

Models A and B trade-off one pair of concerns: model A preserves the existing specification at the expense of allowing an inconsistency in the prototypical cluster-related behaviour of ZWJ; model B, on the other hand, preserves prototypical behaviour of ZWJ in the general Indic model at the expense of introducing the cumbersome sequence < ZWJ, ZWNJ > and deviating slightly from the current specification.

Models A and B both avoid radical departures from the current specification regarding khanda ta, which gives cause to favour these over the other models. Model A does not change the intended specification itself, and requires only relatively minor improvement in the documentation of the specification. A choice to keep model A, therefore, will involve the least work in revising the standard and will avoid breaking implementations designed to work in the way that had been intended. Model B would involve a greater amount of revision to the text of the Standard, as it would require description of intended behaviours in more contexts. Because of the additional behaviors in distinct contexts, it is the more complex model, with greater risk that errors will arise in implementions.

The main case for model B over the existing specification, then, is that of consistency in the prototypical cluster-related behaviour of ZWJ. There are independent exceptions to the prototype, though, so it is not essential that this be preserved (though obviously worthwhile in the absence of any reason to do otherwise). The main case for keeping model A over changing to model B, however, is twofold: it maintains what is already standardized, and it is a simpler model requiring simpler revisions to the documentation with less likelihood for further confusion and errors in impelementations.

In view of these considerations, then, it is proposed that model A be retained as the model for representing khanda ta and other forms of ta in Unicode, and that the problem of the description in the Standard being ambiguous and potentially misleading be resolved by revising the text to clarify the intended behaviour.

Appendix: The significance of cluster-boundaries in text processing

In Indic scripts such as Bengali, the *cluster* or *orthographic syllable* is an important unit of the written structure. Roughly speaking, clusters consist of a base form (usually a consonant) together with various dependent forms written above, below, or to the left or right of the base; the dependent forms might also combine with the base in a ligature.

Clusters are relevant in various aspects of Indic text processing:

- During rendering, the location of re-ordered elements (e.g. reph, i-kaar[2]) is often determined in relation to cluster boundaries. So, for instance, a reph is part of the cluster of a following base and is positioned on that base; an i-kaar matra is positioned on the left side of its cluster.
- The components that comprise a cluster are often combined in a small space, often with vertical staking; as a result, there may not be a clear way to represent the caret within a cluster. Also, due to reordering behaviours of some characters, insertion or deletion of characters anywhere other that at the very end of a cluster can result in significant visual changes that might not be anticipated by users. For these reasons, most implementations do not permit cursor positioning or selection within a cluster.
- Because clusters are (at least typically) graphically self-contained (i.e. typically clusters do not affect other clusters), certain processes such as rendering may be implemented to process text on a cluster-by-cluster basis as a way to achieve certain optimizations.

For these purposes, then, it is important to be able to identify cluster boundaries.

Rendering, drawing of selections, and determining valid character positions are closely related processes, all of which involve the need to identify cluster boundaries at some point. Often, then, a component designed to handle rendering will also handle these other processes as well.

Now, among rendering implementations, there are two overall philosophies that are used. One treats all transformations from a sequence of characters to a sequence of positioned glyphs as a single process, with all of the necessary knowledge maintained in a single resource.[3] This philosophy is exemplified by the Apple Advanced Typography (AAT) and Graphite font technologies, which encapsulate all information about script behaviour and font-specific details inside each font. I will refer to this first philosophy hereafter as "the AAT/Graphite philosophy".

[2] In this discussion, I will assume "i-kaar" refers to a re-ordrant matra, as in the case of Devanagari or Bengali.

[3] This would exclude knowledge of a very general and script-independent nature, such as the Unicode bidirectional algorithm, or how to interpret particular data structures within a font resource.

The other overall philosophy is exemplified by the OpenType font technology, and implementations that use it, such as ICU, Pango and Uniscribe. This philosophy assumes that the knowledge required to display text is of two types: that which relates to the behaviours of a given script, and that which is particular to a given font implementation. For instance, the fact that the Bengali i-kaar takes different initial versus non-initial forms is script-related information that does not need to be duplicated across fonts. In contrast, how a font implements that behaviour (e.g. initial and non-initial i-kaar glyphs versus a single i-kaar glyph and an optional glyph to add the connecting line) is a font-specific detail.

I will refer to this second philosophy hereafter as "the OpenType philosophy".

For implementations that follow the AAT/Graphite philosophy, identification of cluster boundaries is an integral part of the rendering process of transforming strings of characters into sequences of glyphs. Thus, when an application needs to identify cluster boundaries, e.g., for determining valid caret positions, it may need to apply the entire character-to-positioned-glyph transformation, but all of the knowledge related to that transformation is available.

The OpenType philosophy assumes that there are several aspects of text processing that can be done without knowing font-specific details. This is considered more efficient since knowledge is not duplicated across all font resources, and also because certain kinds of calculation can be made without needing to process the *entire* transformation from a string of characters to a sequence of positioned glyphs. Some of the calculations that implementations might do in this way, using script-related knowledge but not font-specific details, include some calculations related to cluster-boundary identification, such as determination of valid caret positions. In effect, these implementations would make such determinations based on character information (the input Unicode character sequence) alone with no reference to glyphs in the selected font.

This difference in philosophies has important implications in relation to consonant clusters in Indic scripts for which a ligated conjunct form may or may not be supported in a font. The presence or absence of a particular conjunct form is significant because, if the conjunct form is supported, then there would be no cluster boundary between the consonants, but a cluster boundary may exist if the conjunct is *not* supported. (So, for instance, a following i-kaar would reorder around a ligated conjunct form; but if the cluster is displayed using a overt-virama form with a cluster boundary between it and the second consonant, the i-kaar would reorder around only the second consonant.)

The point, then, is that implementations that follow the AAT/Graphite philosophy will automatically do the right thing when, say, determining valid caret positions regardless of whether a conjunct is supported in the selected font or not because they already perform such calculations using glyph-related information inside the font. In contrast, implementations that follow the OpenType philosophy and that assume such calculations can be done using character information only face a problem since more information is needed than exists in the string of characters: font-specific details are also needed. For such implementations, additional work will be needed to provide correct behaviour.

This problem that may exist for implementations that follow the OpenType philosophy arises in connection with the Bengali khanda ta under some of the models considered by not others. The point here is that text involving a dead ta and following conjoinable consonant (one with which ta can for a ligated conjunct form) might be displayed using a conjunct form, with no cluster boundary between the ta and the following C, or using khanda ta, with a cluster boundary after the ta. Under models C and D, the khanda ta occurs only and always for specific character sequences, and the cluster boundary following the khanda ta can likewise be determined from the character sequences. In these cases, then, correct determination of cluster boundaries does not depend on font-internal information.

The situation is less simple for models A and B, however. For character sequences using ZWJ/ZWNJ to force a khanda ta, the cluster boundary is predictable from the character sequence. For sequences of the form < ta, virama, C >, the choice of C will affect whether a conjunct or khanda ta is expected. If an implementation could predict which consonants will result in a conjunct and which will not, then it can predict whether a cluster boundary will occur from the character sequence.

The problem, though, is that it *cannot* be predicted which consonants will result in a conjunct and which will not. While there are specific consonants for which conjunct forms would be expected today (t-ta, t-tha, etc.), this set might change over time as the script evolves: new conjunct forms can be innovated (just as khanda ta itself was an innovation), and existing ones can fall out of common use. Moreover, there is still the problem that a given font might not support all of the conjuncts that are currently known.[4]

This issue of cluster-boundary identification is a distinguishing factor between the four models considered, then, in that models A and B may be more difficult for some implementations that follow the OpenType philosophy than would models C and D.

It is important that this factor not be given *too much* weight in comparing the advantages and disadvantages of these models, however. For one thing, the problem should not be insurmountable for OpenType-philosophy implementations as they can access information within a font if needed. Moreover, regardless of which model for khanda ta is adopted, these implementations will still face the general problem that a given font may or may not support a particular conjunct, with the result that correct cluster-boundary identification will, in some cases, depend upon what glyphs are supported within a font. The ta/khanda ta is only one instance of a more general problem.

References

Suniti Kumar Chatterji. 1926. "The Origin and Development of the Bengali Language." Calcutta.

[4] In this regard, it is noteworthy that among descriptions of Bengali script that list ta conjuncts, some list a t-la conjunct while others do not.