Re: **UAX #29 actions in PRI 355**
From: **Mark Davis (recorder)**
Date: **2018-01-25**

The UTC considered the feedback in PRI 355, and came to the following consensus.

General Conclusion

After much discussion, the UTC decided that the aksara support would be best incorporated into CLDR root segmentation, which provides much more flexibility, and allows for more frequent updates. Moreover, it became clear in the feedback that making the proposed changes requires more information about the affected scripts, to ensure that they are properly handled. Correct treatment of those scripts often requires special cases, which are difficult to handle well in UAX #29. They may also need tailoring by language for cases such as Nepali, which would go into language-specific segmentations.

Moving to CLDR allows for the rules to be progressively improved over time, and changes to be incorporated into UAX #29 once a sufficient level of stability is attained.

Action: The changes for aksara in the current proposed draft will be reverted. Instead, strong recommendations to use the rules in CLDR will be added to the draft, and the CLDR/ICU committees are requested to incorporate a limited set of rules into CLDR/ICU root segmentation for GCB. Some changes to the rules and property values should be made before forwarding to CLDR — those are listed below.

Other actions agreed to in discussion are listed in the table below.

Feedback	Actions

Date/Time: Mon Sep 25 10:21:46 CDT 2017 **Name:** Tsuyoshi Ito **Report Type:** Error Report (PRI #355) **Opt Subject:** The regular expressions for a grapheme cluster in Table 1b of UAX #29 do not match the rules in Section 3.1.1 Table 1b of UAX #29, "Unicode Text Segmentation" (http://www.unicode.org/reports/tr29/tr29-31.html), shows the regular expressions for a legacy and an extended grapheme cluster. Section 6.3 seems to indicate that they are supposed to be equivalent to the rules in Section 3.1.1. (However, to be honest, the wording of Section 6.3 is not very clear to me. It says "The conversion into a regular expression is fairly straightforward for the grapheme cluster boundaries of Table 2." but Table 2 is a summary of the Grapheme_Cluster_Break property values, not the rules to determine grapheme cluster boundaries.) However, I think that they are quite different. For example: * According to the rules in Section 3.1.1, a string of more than two regional indicator symbols is not a (legacy or extended) single grapheme cluster. However, according to the regular expressions in Table 1b, it is a single (legacy and extended) grapheme cluster. * According to the rules in Section 3.1.1, an emoji zwj sequence is a single grapheme cluster. However, the regular expression for neither a legacy nor extended grapheme cluster treats ZWJ in a special way, and it puts a grapheme cluster boundary before and after ZWJ. Please consider one of the following changes: Option 1: The regular expressions in Table 1b (and the regular expressions in Table 1c used there) should be updated to match the rules in Section 3.1.1. Option 2: The text of Sections 3.1.1 and 6.3 should be updated to clarify that the regular expressions in Table 1b do not necessarily match the rules in Section 3.1.1.	http://www.unicode.org/reports/tr29/proposed.html#Table_Combining_Char_Sequences_and_Grapheme_Clusters has a revised version. Action: review the new text to make sure the points are covered.
Date/Time: Tue Oct 10 10:06:06 CDT 2017 **Name:** David Corbett **Report Type:** Error Report **Opt Subject:** PRI #355: Cursors in ligatures "For example, the text editing framework must know if a digraph is represented as a single glyph in the font, which therefore cannot have a cursor separating	Action: modify the text to indicate that "may not be able to" instead of "cannot" (Example: if there is not font support for indicating

its two parts.” That is not true: text editing frameworks can and do put cursors within ligature glyphs.	cursor divisions, or if the cursor divisions would require capabilities that the text editing framework doesn’t support, like horizontal cursors in vertically stacked ligatures.)
Date/Time: Tue Oct 10 10:15:49 CDT 2017 **Name:** David Corbett **Report Type:** Public Review Issue **Opt Subject:** PRI #355: Devanagari kshi does not need tailoring Table 1a lists ⟨क्षि⟩ as a tailored grapheme cluster, but it no longer needs tailoring.	Action: add pointer to CLDR
Date/Time: Tue Oct 10 10:29:29 CDT 2017 **Name:** David Corbett **Report Type:** Error Report **Opt Subject:** PRI #355: More LinkingConsonants Grapheme_Cluster_Break=LinkingConsonant should be expanded to include Indic_Syllabic_Category=Vowel_Independent and Indic_Syllabic_Category=Consonant_Dead. Independent vowels may be subjoined in Khmer, and Bengali’s khanda ta may take a repha.	Action: Make change to LinkingConsonant, and add review note about the change to broaden the scope of LinkingConsonant, before forwarding to CLDR.
Date/Time: Wed Oct 11 14:18:02 CDT 2017 **Name:** David Corbett **Report Type:** Public Review Issue **Opt Subject:** PRI #355: Indic clusters without virama Some Indic consonant clusters do not use a virama. GB9c should be (StackingConsonant | Virama | ZWJ) × LinkingConsonant, where StackingConsonant is Indic_Syllabic_Category = Consonant_With_Stacker.	Action: Make change, before forwarding to CLDR.
Date/Time: Wed Oct 11 14:33:49 CDT 2017 **Name:** David Corbett **Report Type:** Public Review Issue **Opt Subject:** PRI #355: U+0BCD TAMIL SIGN VIRAMA U+0BCD TAMIL SIGN VIRAMA generally does not create conjuncts. The exceptions are ⟨க்ஷ⟩ and ⟨◌்ர⟩. It may better match user expectations to exclude U+0BCD from GCB = Virama.	See **general discussion** above.

Date/Time: Thu Oct 19 17:50:25 CDT 2017
Name: Roozbeh Pournader
Report Type: Public Review Issue
Opt Subject: Virama and UAX #29

Because of the dual usage of the characters with the Indic Syllabic Category of Virama. It appears to me that a virama is more frequently just a visible killer instead of an invisible stacker.

Tamil is a common example where the visible killer frequency is much higher than the invisible stacker frequency. But I expect several other scripts would have a similar situation, and even for languages such as Hindi, the frequency of visible killer usage would be too high for always disallowing a grapheme break.

If breaks after this class are being forbidden, I suggest removing the Indic_Syllabic_Category = Virama class from the new virama class, and renaming the class to InvisibleStacker.

Also, forbidding breaks between ZWJ and LinkingConsonant appears incorrect. ZWJ is generally used in Indic as an invisible letter. So in that usage, it could be thought of as ending a cluster with a break allowed after it. Also note that ZWJ is used after virama in the legacy representation of Malayalam Chillus, which is still very common on the internet and in newly created content. Forbidding a cluster break between a ZWJ and a consonant would be incorrect in such usage.

Finally, note that it's not just character of InSC=Consonant that take post-stacker forms. Independent Vowels, Consonant Placeholders, and perhaps Consonant_Deads and Consonant_With_Stacker may appear after stackers. There may even be more.

Altogether, I think the proposed rules are based on a simplified version of the Indic grapheme cluster patterns which needs much more research. They should be rewritten to only discourage breaks in these non-controversial cases:

1. Forbid grapheme breaks after all characters of InSC=Invisible_Stacker, regardless of the character that comes after. (We don't need to worry about odd cases, like when an Invisible_Stacker is followed by a space or punctuation. These are malformed text, and it's OK to go either way on malformed text.)

See **general discussion** above.

The test files from the Indic government include cases where the virama is a visible killer, and does not otherwise normally affect layout.

Recommend to CLDR to initially restrict the scripts of the GCB = Virama and GCB = LinkingConsonant to the scripts for which India supplies comprehensive test files, pending review. For example, GCB=Virama could be

Indic_Syllabic_Category = Virama, or
Indic_Syllabic_Category = Invisible_Stacker
and not General_Category = Spacing_Mark
and Script = Devanagari, or
Script = Bengali

2. Forbid breaks before all InSC={Virama, Invisible_Stacker, Pure_Killer} (note that this is already the case, since they are currently categorized as Extend, but may be necessary if the Extend class is split).	
Date/Time: Sat Dec 9 18:16:56 CST 2017 **Name:** Richard Wordingham **Report Type:** Public Review Issue **Opt Subject:** PRI 355: Proposed Update UAX #29 Unicode Text Segmentation If the second of the three notes at the end of Section 3.1.1 (starting "A tailoring for basic aksara support") is to be retained, note that it will typically be untrue for a language with both a virama or an invisible_stacker and a pure killer, e.g. U+1039 MYANMAR SIGN VIRAMA and U+103A MYANMAR SIGN ASAT and U+0D4D MALAYALAM SIGN VIRAMA and U+0D3B MALAYALAM SIGN VERTICAL BAR VIRAMA and U+0D3C MALAYALAM SIGN CIRCULAR VIRAMA. The note also assumes that the tailoring is specific to a script. Further to the point about cursors stepping through ligatures, this can be seen with the Latin ligature 'ffi' and the Arabic lam-alif. One can also find the cursor stepping through, sometimes not visibly, through Indic aksharas composed of multiple base consonants and even the Tai Tham ligature. Is there evidence to support the claim in Section 3, 'The extended grapheme clusters should be used in implementations in preference to legacy grapheme clusters, because they provide better results for Indic scripts such as Tamil or Devanagari in which editing by orthographic syllable is typically preferred.' Further more, why should the preferences of Indians determine how Cambodians edit. The difference between the two types of cluster will become larger when extended grapheme clusters grow to be whole aksharas (with limited exemptions for Ahom, Myanmar and Tai Tham), and modifying or inserting a character required deleting all the akshara's character from its position onwards.	Actions: Revise the note in line with the **general discussion** to point to CLDR. • A tailoring for basic *aksara* support would add … The cursor change is listed above. This is more reasons to limit the number of scripts initially.
Date/Time: Mon Dec 11 12:35:36 CST 2017 **Name:** Otto Stolz **Report Type:** Error Report **Opt Subject:** Proposed Update Unicode® Standard Annex #29 http://www.unicode.org/reports/tr29/tr29-32.html#Word_Boundaries	Action: change to “ignoring words that contain only whitespace, punctuation, and similar characters”

Figure 2 does not match the pertinent text which says: “That is done with the above boundaries by ignoring any words that do not contain a letter, as in Figure 2.” In contrast, figure 2 comprises the word “32.3” that does not contain any letter.	
Date/Time: Tue Jan 2 08:14:48 CST 2018 **Name:** Manish Goregaokar **Report Type:** Error Report **Opt Subject:** UAX #29: Eliminating E_Modifier Originally discussed at https://unicode.org/mail-arch/unicode-ml/y2018-m01/0000.html In UAX 29, the GB10 rule[1] (and the WB14 rule[2]) states that we should not break before E_modifier characters in case it is after an emoji base (with optional Extend characters in between) Given that the spec is allowed to ignore degenerates, we should merge E_Modifier into Extend, as outlined in Mark's email[4], and eliminate GB10 entirely. <random non-emoji, skin tone modifier> sounds very much like a degenerate case to me. lt;non-EBG GAZ emoji, skin tone> also feels rather degenerate. There are only three GAZes (heart (U+2764), kiss (U+1F48B), speech bubble (U+1F5E8)) and I can't see why you'd end up with a skin tone modifier on them except by accident. Additionally, the current draft[3] is eliminating the GAZ categories anyway. Thanks, -Manish [1]: http://www.unicode.org/reports/tr29/#GB10 [2]: http://www.unicode.org/reports/tr29/#WB14 [3]: https://www.unicode.org/reports/tr29/tr29-32.html [4]: https://unicode.org/mail-arch/unicode-ml/y2018-m01/0004.html	Action: do what is proposed. (It simplifies the expressions, and the additional degenerate cases that it allows don’t matter much in practice.) That is: 1. Move E_Modifier characters into Extend 2. Mark E_Modifier and E_Base characters as “This value is obsolete and unused.” 3. Remove G10.
Date/Time: Mon Jan 22 21:04:53 CST 2018 **Name:** Richard Wordingham **Report Type:** Public Review Issue	Actions: Change paragraph to have “default”.

Opt Subject: PRI355: Akshara Boundaries
The major principled issue I have is that UAX#29 can no longer claim to have a sound definition of the concept of a 'user-perceived character'. Perhaps it never did.

Some of the claims would be better if there were evidence to back them up. For example, this evening I did a quick bit of research and asked the Korean owner of the local Korean restaurant how many letters there were in the hangul spelling of 'Gangnam'. She traced out the spelling of the word (강남) and came back with the answer '6'. UAX#29 claims it has 2 user-perceived characters. You might also argue that she has spent too long in England to be a useful informant.

The following old paragraph causes grief for me:

"As far as a user is concerned, the underlying representation of text is not important, but it is important that an editing interface present a uniform implementation of what the user thinks of as characters. Grapheme clusters commonly behave as units in terms of mouse selection, arrow key movement, backspacing, and so on. For example, when a grapheme cluster is represented internally by a character sequence consisting of base character + accents, then using the right arrow key would skip from the start of the base character to the end of the last accent."

The problem is that many editors read this as saying that the arrow keys should move by whole characters. The result of this is that in many applications, to replace the first character of a grapheme cluster one must retype the entire grapheme cluster. With a grapheme cluster of three characters, as is common in Thai and Korean, this is irritating. With a grapheme cluster of four or five characters, as is common in Northern Thai, it is annoying.

The prospect of the grapheme cluster being extended to include a whole akshara fills me with dismay. Consider the Northern Thai word □□□□ <U+1A49 HIGH HA, U+1A60 SAKOT, U+1A3E MA, U+1A70 SIGN OO, U+1A6C SIGN OA BELOW, U+1A6B SIGN O, U+1A61 SIGN A> /mɔʔ/ 'scrumptious'. At present, this 7 character word is split into three grapheme clusters, of lengths 2, 4 and 1. However, it is clearly a single akshara. To change the first character, I would have to also retype the other 6 characters.

Grapheme clusters can be treated as default units in terms of mouse selection, arrow key movement, backspacing, drop-caps, and so on.

Add paragraph:

Similarly, arrow keys could be tailored by language, or use knowledge of particular fonts to move more granularly, where it is useful to be able to edit individual components, such as in Northern Thai.

For the other comments, see the **general discussion**.

My first thought that changing software that way would breach the UK's Equality Act 2010, by further restricting the ability of Northern Thai users to do character by character editing. (My wife's protected characteristic extends to me for the purposes of the Act.) However, there may be a get-out in the form of Schedule 3 Section 30 (https://www.legislation.gov.uk/ukpga/2010/15/schedule/3/paragraph/30). The supplier of the service can claim that they only supply a character by character editing facility to the ethnic groups using simple scripts, and that they are under no obligation to supply the service to members of other ethnic groups. -
"If a service is generally provided only for persons who share a protected characteristic, a person (A) who normally provides the service for persons who share that characteristic does not contravene section 29(1) or (2)—

(a)by insisting on providing the service in the way A normally provides it, or

(b)if A reasonably thinks it is impracticable to provide the service to persons who do not share that characteristic, by refusing to provide the service."

But what an embarrassing defence to offer!

However, there is another reason for rejecting the extension of grapheme clusters to whole aksharas. Currently, U+1A63 TAI THAM VOWEL SIGN AA starts a grapheme cluster. However, for non-defective text, it is part of the same akshara as the preceding grapheme cluster. Now, the decision to make U+1A63 start a new grapheme cluster is intrinsically reasonable. It can have its own stack with a subscript consonant and even a vowel, and it is not difficult to find manuscripts showing a line break before it, e.g. L2/07-007 Figure 9b Leaf 2 lines 2/3, □□□□□□□-□□□□.

I believe that the akshara should be a level of text above the grapheme cluster. Ideally, it would be below the level of a word, but of course in Sanskrit, word boundaries readily occur within present day grapheme clusters. (I made this recommendation in L2/17-122.)

Further comments apply to the definition of akshara boundaries, regardless of whether they are to coincide with the boundaries of

grapheme clusters.

These rules do not work well where virama may fall back to visible virama. This is particularly the case with Tamil, where conjuncts are restricted to K.SSA and SH.RII. Johny Cibu provided an example where the title துக்ளக் is broken as [ta-u, ka-virama, lla, ka-virama]. However, as per the proposed algorithm it would be: [ta-u, ka-virama-lla, ka-virama]

http://www.chennaispider.com/attachments/Resources/3486-7144-Thuglak-Tamil-Magazine-Chennai.jpg

For native intuition, I would cite the Tamil letter-counting account at https://venkatarangan.com/blog/content/binary/Counting%20Letters%20in%20an%20Unicode%20String.pdf .
What the author counts is not spacing glyphs, but vowel letters and consonant characters, with two significant modifications. Firstly, K.SSA counts as just one consonant, and SH.R.II is also counted as containing a single consonant. In other words, the Tamil virama character works as a pure killer except in those two environments. This is also the story the TUNE protagonists tell us. It will be an inelegant rule for UAX#29, but, unfortunately, reality is messy.

To quote Johny Cibu further:

"Malayalam could be a similar story. In case of Malayalam, it can be font specific because of the existence of traditional and reformed writing styles. A conjunct might be a ligature in traditional; and it might get displayed with explicit virama in the reformed style. For example see the poster with word ഉസ്താദ് broken as [u, sa-virama, ta-aa, da-virama]
- as it is written in the reformed style. As per the proposed algorithm, it would be [u, sa-virama-ta-aa, da-virama]. These breaks would be used by the traditional style of writing.

https://upload.wikimedia.org/wikipedia/en/6/64/Ustad_Hotel_%282012%29_-_Poster.jpg

I believe there is a problem with the first two examples in Table 12-33. If one suffixed <U+0D15 MALAYALAM LETTER KA, U+0D3E MALAYALAM VOWEL SIGN AA> to the first two examples, yielding *പാലുക്കാ and

*എ്ന്നാകാ, one would have three Malayalam aksharas, not two extended grapheme clusters as the proposed rules would say.	