

UCA: Problems with discontinuous contractions

Author: Markus Scherer

Date: 2012-mar-15

This document lays out several problems with handling discontinuous contractions as specified in the Unicode Collation Algorithm (UCA, [UTS #10](#)). It then proposes several possible options for dealing with these problems. The exact changes to the UCA specification and the Default Unicode Collation Element Table (DUCET) are indicated with each option.

In the current UCA+DUCET, only three contractions, each for a Tibetan composite vowel for Sanskrit transcriptions, require the most complex processing. The CLDR Myanmar tailoring adds one more problematic contraction.

The goal is to limit the complexity necessary in conformant implementations of the UCA that do not modify an input Unicode string while mapping it to a sequence of collation elements.

Problem: Discontinuous contraction + contraction starting with non-zero combining mark

A contraction gives a particular sequence of characters different sorting behavior than the isolated processing of each of those characters. ([UCA section 3.3.2 Contractions](#)) For example, according to the DUCET, U+0FB2 (Tibetan Subjoined RA) plus U+0F71 (Vowel Sign AA) plus U+0F80 (Vowel Sign Reversed I) together yield a different sequence of collation elements (CEs) than the CEs for each of the characters in isolation.

A discontinuous contraction is a contraction with a trailing non-zero combining mark, such as **0FB2+0F71+0F80**. Similar to NFC normalization, an intervening non-zero combining mark must be skipped during the contraction matching. For example, for a string of **<0FB2, 0F71, 0F71, 0F80>** the discontinuous contraction of 0FB2+0F71+0F80 must be matched, and the result is CE(0FB2+0F71+0F80), CE(0F71).

The [UCA spec. section 4 Main Algorithm. Step 2.1](#), specifies that this is done by rewriting the input string, moving skipped combining marks behind matching ones. For example:

- Input string: **<0FB2, 0F71, 0F71, 0F80>**
- Rewrite to: **<0FB2, 0F71, 0F80, 0F71>**
- Result: CE(0FB2+0F71+0F80), CE(0F71)

It is possible that a contraction can begin with a non-zero combining mark, and that such a character is skipped during discontinuous-contraction matching. For example, the DUCET includes contractions 0FB2+0F71+0F80 as well as 0F71+0F74, and all of 0F71, 0F74 and 0F80 are non-zero combining marks with different combining classes. For example:

- Input string: **<0FB2, 0F71, 0F71, 0F80, 0F74>**
- Rewrite to: **<0FB2, 0F71, 0F80, 0F71, 0F74>**
- Result: CE(0FB2+0F71+0F80), CE(0F71+0F74)

For performance, it is important to avoid rewriting the string. For example, the skipped combining marks might be collected in a temporary buffer and their collation elements appended to those for the contraction. However, in the example above, the temporary buffer would contain only 0F71 but not 0F74. Finding the contraction of 0F71+0F74 across the boundaries of the temporary buffer adds significant complexity to the code.

In particular, it is possible that a discontinuous contraction starts in the temporary buffer, skips combining marks from the buffer, and skips and matches several characters in the remainder of the input string.

It is also possible that a “normal” (contiguous) contraction starts in the temporary buffer and continues into the remainder of the input string, including characters with ccc=0. For example:

- DUCET contraction: 0430+0308
- CLDR Myanmar contraction: 1037+1038
- Input string: <0430, 1037, 0308, 1038>
- Result: CE(0430+0308), CE(1037, 1038).

In summary, this problem arises from the discontinuous-contraction-matching part of the UCA spec, together with contractions starting with non-zero combining marks, when two such contractions occur together and are interleaved in canonical order.

Problem: Discontiguous contraction + contraction with trailing combining mark of a composite

It is possible for a Collation Element Table to define a contraction that ends with a combining mark, and that combining mark is the second character in the canonical decomposition of some character. For example, the contraction of 0FB2+0F71+0F80 vs. the canonical decomposition of 0F81 to <0F71, 0F80>.

For performance, it is important that an implementation normally avoid normalization (see the [UCA spec, section 6.5 Avoiding Normalization](#)). Most strings are canonically ordered but not necessarily decomposed, which means they are in form “Fast C or D” ([FCD](#)).

Given such an implementation that works on FCD input strings, the contraction **0FB2+0F71+0F80**, and an input string of <0FB2, 0F71, 0F81>, the implementation would fail to detect the contraction although it would find it in the NFD form of the string. In NFD, the match involves discontinuous-contraction matching.

This problem arises from the discontinuous-contraction-matching part of the UCA spec, together with characters that have canonical decompositions to sequences of different-ccc non-zero combining marks (which in Unicode 6.1 is exactly the three Tibetan composite vowel signs mentioned in the Scope section below), together with a collation implementation that accepts FCD input strings. This problem is independent of whether contractions like 0F71+0F72 exist.

Problem: Discontiguous contraction + missing prefix contraction

The UCA Main Algorithm fails to match some contractions that intuitively seem intended to be matched.

Consider a contraction of $A+n+q$ and an input string of $Ampnq$, with m, n, p, q all non-zero combining marks with different ccc in canonical order. Note that $A+n$ is not a contraction.

- S2.1 finds the initial A .
- S2.1.1 starts a loop over the following combining marks.
- S2.1.2: m is not blocked from A but there is no match for $A+m$
- S2.1.3: Loop.
 - The Algorithm does not explicitly specify to loop back to S2.1.1 but this can be inferred from the statement of S2.1.1.
- S2.1.2: n is not blocked from A but there is no (complete) match for $A+n$
- loop
- same for $A+p$
- same for $A+q$
- no contraction match $\rightarrow CE(Ampnq) = CE(A), CE(m), CE(n), CE(p), CE(q)$

However, intuitively, it seems the result should be $CE(A+n+q), CE(m), CE(p)$.

Also, it is unclear what S2.1 "Find the longest initial substring S at each point that has a match in the table." means when there is a contraction of $A+B+n$ where $ccc(B)=0$, but there is no contraction of $A+B$, and given the input string $ABmn$. Strictly speaking, the "longest initial substring ... that has a match in the table" is A because $A+B$ is not in the table. The algorithm will not attempt a discontinuous-contraction match because the following character B has $ccc=0$. Intuitively, it seems the result should be $CE(A+B+n), CE(m)$.

It would be possible to achieve the intuitive results by making the algorithm (and implementations) more complex: The algorithm would have to continue looping for partial matches and track which combining marks were skipped vs. which ones partially matched. When there is a full match, only the matching marks would be consumed, leaving the skipped ones. When a partial match eventually fails, the state from after the last match must be restored. This would add even more complexity especially to handling contractions that start from skipped combining marks.

The proposals below are for simpler options.

Scope

In the DUCET, there are only three contractions that start with non-zero combining marks. They are contractions of Tibetan vowel signs, and each of these contractions is canonically equivalent to a composite vowel sign: $0F71+0F72=0F73$, $0F71+0F74=0F75$, and $0F71+0F80=0F81$.

(These contractions are some of the Tibetan compound vowels for Sanskrit transcription. The composite characters are annotated in the Unicode code charts as "discouraged".)

UCA+DUCET yield the following sort order, with primary-level differences between each string:

| | |
|--------------------|--------------------------|
| ... | |
| 0F71 | (Vowel Sign AA) |
| 0F72 | (Vowel Sign I) |
| $0F71+0F72 = 0F73$ | (Vowel Sign II) |
| 0F80 | (Vowel Sign Reversed I) |
| $0F71+0F80 = 0F81$ | (Vowel Sign Reversed II) |

| | |
|------------------|------------------------|
| 0F74 | (Vowel Sign U) |
| 0F71+0F74 = 0F75 | (Vowel Sign UU) |
| 0F76 | (Vowel Sign Vocalic R) |
| ... | |

The CLDR Myanmar tailoring contains one contraction of 1037+1038 (Myanmar Dot Below+Visarga) where $\text{ccc}(1037)=7$ and $\text{ccc}(1038)=0$.

The CLDR Lithuanian tailoring contains five contractions starting with 0307 Dot Above where all contraction characters have the same $\text{ccc}=230$. They do not separate (skipped vs. after-the-match) in discontinuous contractions because their leading characters would be Blocked, or be Blocking another character. However, it is possible that both of the combining marks are skipped in discontinuous contraction and that their contraction would then need to be detected. This would require contraction matching inside a skipped-marks buffer but would not require rewriting the input string.

- These contractions are: 0307+0300, 0307+0340, 0307+0301, 0307+0341, 0307+0303

In other words, only the three DUCET Tibetan contractions and one CLDR Myanmar contraction require a complex rewrite of the input string. This complexity is costly in terms of performance, code size and reliability.

The DUCET appears to not contain examples of missing-prefix-contractions, but some of the CLDR tailorings do. For example, the CLDR Croatia tailoring has a contraction of 0064+007A+030C but not one of 0064+007A.

Proposal

Abolish discontinuous-contraction matching.

- No change to the DUCET or to CLDR
 - Except if there are input strings that currently benefit from discontinuous-contraction matching and which would sort differently; such sequences could be added as additional contractions.
- Change the UCA:
 - In section 3.3.2 Contractions, remove the requirement to match contractions while skipping combining marks.
 - In section 4 Main Algorithm, remove sub-steps 2.1.1, 2.1.2, 2.1.3.
- Impact:
 - For any contraction with trailing non-zero combining marks, if an input string contains an intervening non-zero combining mark with a lower combining class, then the contraction will not be detected. If such an input sequence is important, then the sequence including the intervening combining mark needs to be added as another contraction. (See the second half of [UCA section 3.3.2 Contractions](#).)
 - The UCA 6.1 DUCET contains 78 contractions that end with non-zero combining marks. Most of them are contractions of one Cyrillic letter with one accent-above mark, e.g., Cyrillic a+Diaeresis. CLDR tailorings contain many similar contractions.
 - Implementations would only detect contiguous contraction matches, removing significant amounts of code and complexity.

If it is acceptable to remove discontinuous-contraction matching, then the main source of the above problems disappears and none of the following alternatives are relevant.

Alternative Proposal

Require prefix contractions

- No change to the DUCET
- Change the UCA:
 - In section 3.7 Well-Formed Collation Element Tables, require that, for any contraction, every prefix string must also be a contraction.
- Change CLDR:
 - Add missing prefix contractions.
- Impact:
 - Table builders report errors when prefix contractions are missing.
 - Implementations detect contractions as expected.
 - Implementations need not track partial matches nor back up to after the last match.

Note: It would not be correct to just auto-generate missing prefix contractions. If a user tailors A+B+n as well as B+C, and if then A+B is auto-generated resulting in (CE(A), CE(B)), then the user would be surprised that CE(ABC) ≠ CE(A), CE(B+C).

Options for contractions that start with non-zero combining marks

We propose adding limitations on collation contractions used and allowed in the UCA. We see a few options, starting with the most desirable one from an implementer's perspective.

Option 1

Forbid contractions that start with non-zero combining marks.

- Change the DUCET:
 - Remove the three problematic contractions, and change the mappings for the canonically equivalent 0F73, 0F75 and 0F81 to the then-corresponding expansions (e.g., CE(0F73)=CE(0F71), CE(0F72)).
- Change the UCA:
 - In section 3 Collation Element Table, specify that the first character of a contraction must have ccc=0, or else all ccc must be the same.
 - In section 4 Main Algorithm, note this limitation.
- Change CLDR:
 - Remove the problematic Myanmar contraction and the five Lithuanian contractions with Dot Above.
- Impact:
 - In the modified UCA+DUCET, the strings <0F71, 0F72>, <0F71, 0F80> and <0F71, 0F74>, and their canonically equivalent composites (vowel signs II, Reversed II, and UU), would sort between 0F71 (AA) and 0F72 (I), rather than one or more places after 0F72 as is currently the case.
 - Tailorings would not be able to add such problematic contractions if the collation implementation is bound by this same limitation.
 - Implementations would simply look up collation elements for each skipped

combining mark in isolation. They could rely on such characters not having contraction data.

Option 2

Forbid contractions that start with non-zero combining marks and contain characters with different ccc values.

- Change the DUCET:
 - Remove the three problematic contractions, and change the mappings for the canonically equivalent 0F73, 0F75 and 0F81 to the then-corresponding expansions (e.g., CE(0F73)=CE(0F71), CE(0F72)).
- Change the UCA:
 - In section 3 Collation Element Table, specify that the first character of a contraction must have ccc=0, or else all ccc must be the same.
 - In section 4 Main Algorithm, note this limitation.
- Change CLDR:
 - Remove the problematic Myanmar contraction.
- Impact:
 - In the modified UCA+DUCET, the strings <0F71, 0F72>, <0F71, 0F80> and <0F71, 0F74>, and their canonically equivalent composites (vowel signs II, Reversed II, and UU), would sort between 0F71 (AA) and 0F72 (I), rather than one or more places after 0F72 as is currently the case.
 - Tailorings would not be able to add such problematic contractions if the collation implementation is bound by this same limitation.
 - Implementations would fetch collation elements for a sequence of skipped combining marks, including possible contractions within this sequence (e.g., via recursion), but without having to match contraction suffixes beyond the sequence of skipped combining marks.

Option 3

Ignore contractions that start with non-zero combining marks when they were skipped in discontinuous-contraction matching.

- No change to the DUCET or to CLDR
- Change the UCA:
 - In section 4 Main Algorithm, specify that for combining marks skipped in Step 2.1, collation elements are fetched by looking up each skipped character in isolation.
- Impact:
 - In the modified UCA+DUCET, strings like <0F71, 0F74> sort like in UCA 6.1. However, a string like <**0FB2**, **0F71**, 0F71, **0F80**, 0F74> would sort differently, as CE(0FB2+0F71+0F80), CE(0F71), CE(0F74) rather than CE(0FB2+0F71+0F80), CE(0F71+0F74).
 - Implementations would ignore contraction data for combining marks that are skipped in discontinuous-contraction matching and always fetch the collation elements for each skipped combining mark in isolation. Combining marks that are found in normal processing, rather than skipped in discontinuous contraction, are processed normally, including possible contraction matching.

Option 4

Forbid contractions that start with non-zero combining marks and contain ccc=0 characters.

- No change to the DUCET
- Change the UCA:
 - In section 3 Collation Element Table, specify that if the first character of a contraction has ccc≠0 then every character in the contraction must have ccc≠0.
- Change CLDR:
 - Remove the problematic Myanmar contraction.
- Impact:
 - No change in UCA+DUCET behavior.
 - Tailorings would not be able to add contractions like 0300+0041 (Grave+A) or 1037+1038 (Myanmar Dot Below+Visarga) if the collation implementation is bound by this same limitation.
 - Implementations would be significantly more complex than for the other options, but with some limits. An implementation of this option could write the skipped combining marks to a temporary buffer and then append all of the non-zero combining marks that *follow* a discontinuous contraction to that buffer. It would have to recursively process the characters in that buffer, including contractions, and append the resulting collation elements to those from the original discontinuous contraction. This new UCA limitation ensures that the buffer contains the entire context for any nested contraction, and that contraction matching need not extend to or beyond any following character with ccc=0. (It might be possible to replace the recursive processing of the temporary buffer with a loop that works like UCA Step 2.1, successively modifying the buffer string.)

Option 5

No change. Handling discontinuous contractions together with contractions that start with non-zero combining marks is complex but ultimately doable.