L2/15-108

Re: **Collation problems in the draft tr10 text**
From: **Mark Davis**
Date: **2015-05-01**
Link: **https://goo.gl/D0sbue**

There are some errors in tr10 that need to be address

http://www.unicode.org/reports/tr10/proposed.html#S1.1

Current

- Conformant implementations may skip this step in certain circumstances: see Section 6.5, Avoiding Normalization for more information.

Proposed

- Conformant implementations may skip this step in certain circumstances, as long as they get the same results. For techniques that may be useful in such an approach, see Section 6.5, Avoiding Normalization.

Discussion.

Make it clearer that the same results are required.

http://www.unicode.org/reports/tr10/proposed.html#S2.1.2

7.0 text:

Note: A non-starter in a string is called blocked if there is another non-starter of the same canonical combining class or zero between it and the last character of canonical combining class 0.

8.0 draft text:

Note: The non-starter C is blocked from S if there is another character B between S and C, and either B has canonical combining class zero (ccc=0), or ccc(B) >= ccc(C).

Add:

Note: this condition is is specific to non-starters, and not precisely the same as in normalization, since it is dealing with discontiguous-contraction, not normalization forms. Jamos and other starters are only supported with contiguous contractions.

Discussion.

We should also add a note, because we've had at least one person mix these up.

http://www.unicode.org/reports/tr10/proposed.html#Avoiding_Normalization

Current

Characters with canonical decompositions do not require mappings to collation elements, because S1.1 maps them to collation elements based upon their decompositions. However, they may be given mappings to collation elements anyway. The weights in those collation elements must be computed in such a way that they will sort in the same relative location as if the characters were decomposed using Normalization Form D. Including these mappings allows an implementation handling a restricted repertoire of supported characters to compare strings correctly without performing the normalization in S1.1 of the algorithm. It is recommended that implementations correctly sort all strings that are in the format known as "Fast C or D form" (FCD) even if normalization is off, because this permits more efficient sorting for locales whose customary characters do not use multiple combining marks. For more information on FCD, see [UTN5].

Proposed

Conformant implementations do not have to normalize text in Section 4.1 [#Step_1], as long as they are structured so as to get the same results as if they did.

In the normal algorithm, characters with canonical decompositions do not require mappings to collation elements, because S1.1 maps them to collation elements based upon their decompositions. So there doesn't need to be a mapping for ü, because both "u" and "̈" will have mappings. However, these characters may also be given mappings to a sequence of collation elements. Including such mappings allows an implementation compare strings correctly without performing the normalization in S1.1 of the algorithm, for the vast majority of text.

While such an approach can have significantly improved performance, there are various issues that need to be handled, including but not limited to the following:

1. Typically the easiest way to manage the data is to add mappings for all the canonically equivalent strings, the so-called "canonical closure". Thus each of {ǭ, ǫ +̄ , ō +̨ , o+̄ +̨ , o+̨ , +̄ } can map to the same weights.
2. The weights in those collation elements must be computed in such a way that they will sort in the same relative location as if the characters were decomposed using Normalization Form D.
3. The easiest approach is to detected sequences that are in the format known as "Fast C or D form" (FCD: see [UTN5]), and handle them with the additional weights.
4. In any difficult cases, such as if a sequence is not in FCD form, or there are contractions that cross sequence boundaries, the algorithm can fall back to doing a full NFD normalization.

Discussion.

The original text was not specific enough, and also was phrased in a way that looked like it was a more complete solution rather than guidelines.

http://www.unicode.org/reports/tr10/proposed.html#Forcing_Stable_Sorts

7.0 text:

Forcing stable results can be done by appending the current record number to the strings to be compared. (The implementation may not actually append the number; it may use some other mechanism, but the effect would be the same.)

8.0 draft text:

A non-stable sort algorithm can be forced to produce stable results by comparing the current record number (or some other value that is guaranteed to be unique for each record) for otherwise equal strings.

Discussion.

It is insufficient to have "some other value that is guaranteed to be unique". Uniqueness does not mean stability. "A stable sort is one where two records with a field that compares as equal will retain their order if sorted according to that field." It *must* be a value that is monotonically increasing with the record number, so that the original order is preserved with equal keys.

So the parenthetical must be removed.