Proposed Recommendations Regarding Certain Variants in Devanagari LGR

DATE: 2018-06-28

# Summary

The most commonly defined variants are those that substitute single code points and where neither the code points nor the resulting labels are subject to code point context rules or whole-label rules, respectively. In cases where n:m variants are defined (mapping code point sequences of length n to code point sequences of length m), additional complications may arise if n and m share some common code points. In such cases, a variant context rule may need to be defined on the variant so it is only defined in situations where the substitution is valid. Otherwise, the resulting sets of variant labels are either not transitive and symmetric, or they present difficulties in efficient computation of index variants, an essential tool to quickly compute collisions between variant labels. The remainder of this document discusses these issues and how they relate to the Devanagari LGR proposal of 2018-05-20.

# Background

In order to efficiently detect whether a label is blocked by a variant label, one normally computes a so-called “index variant” for both and if they are equal, the two labels are variants of each other. If one has a list of index variants for all registered labels, an application for a new label can be very quickly checked for collisions, as long as the computation of the index variant itself is efficient. To ensure efficient calculation under certain variant set definitions, it is important to be able to calculate the index variant in a single pass (as described below) and still get a correct result.

## Requirements for Index Variants

For the Index Variant method to work, the space of all labels and their variant labels must be divisible into *variant label sets* so that

1. any label and all its variants belong to the same set
2. no two sets overlap
3. all labels in the set generate the same index variant

If these conditions are met, two labels with the same index variant are variants of each other.

For these requirements, it is inessential whether any enumerated variants are also valid labels or not, as long as any invalid labels also belong to only one set.

## Generating Index Variants

Index variant generation starts with a valid label. (There appears to be no benefit in going to any extra length in making LGRs produce predictable index variant results for invalid labels; however, if doing so produces an LGR set that can be more easily verified as being correct, there's no reason not to.)

Index variant generation proceeds left-to-right in code point sequence. At each point, for any code point or sequence for which a variant is defined, the lowest variant in code point order is substituted (or the original code point or sequence retained if lowest or without variant). If more than one code point/code point sequence start at a give point, an index variant candidate is calculated for each case and the processing continues for that candidate at the end of the given sequence. Each division into sequences is called a *partition* and an index variant candidate is produced for each possible partition of the given label. In determining available variants, any variant that has a *variant* context rule and does not satisfy the rule is ignored. At the end, the lowest candidate becomes the Index Variant. If two variants are such that one is a prefix of another, the shorter variant (i.e. the prefix) becomes the Index Variant.

Whether or not an index variant is a valid label does not matter. In fact, it would be cost prohibitive to insist on index variants to be valid labels: the only way to guarantee that in the general case would be to enumerate all variants and at the end pick the lowest. Many labels have thousands of possible blocked variants. Therefore, index variant generation ignores any *code point* context rules or whole-label rules.

Note that for the Root Zone, index variants are computed based on the merged or "common" LGR, therefore "mixed script" labels are notionally in-repertoire.

## Transitivity of Code Point Variant Sets and Variant Label Sets

Transitivity means that all variants in the set are variants of each other. See RFC 8228 for a discussion of this and other concepts related to variants.

For enumerating variants it is strictly required that all allocatable variants form a fully transitive variant label set, so that the same set of variants is generated no matter which of the variants is the starting label. For checking collisions, we do not want to have to enumerate all blocked variants – doing so is prohibitive in terms of performance. Therefore, we only require that an LGR is well-behaved as far as index variant calculation is concerned.

When *code point* variant sets are defined for code point sequences in LGRs where subsequences of the same sequences are part of the LGR's repertoire (and especially, if they have variants in their own right) then a variant label set may not be transitive, or non-overlapping, even if the code point variant set is defined in a formally transitive manner.

Any LGR with such overlapping sequences requires special attention to ensure that it is well-behaved.

# Full Analysis of Kashmiri Vowel 0973

(The corresponding variant definitions for Kashmiri matras, such as 093B 🡪 093E 0902 have similar issues and require similar solutions.)

In the following analysis, we assume that these mappings have been defined for blocked variants:

|  |  |
| --- | --- |
| Variant Set 1 | 0902 🡨🡪 093A0902 🡨🡪 0A02**093A 🡨🡪 0A02** |
| Variant Set 2 | 0905 0902 🡨🡪 0973 |

The entry 093A 🡨 🡪 0A02 (shown in **bold**) is needed to make the *code point* Variant Set 1 regular, that is symmetric and transitive. The other *code point* variant set with two elements is trivially transitive. All elements in either set can reach all other elements in the same *code point* variant set. Having a formally transitive code point variant set is at once useful to ensure that the LGR is regular as far as any 1:1 variant mappings are concerned, and, at the same time, 093A and 0A02 appear to be just as related as 0902 and 0A02.

However, even with both sets transitive on their own, nothing fixes the fact that the two sets are related because they share a "common suffix" sequence (0902 is a suffix of 0902 and 0905 0902). We will see that variant *label* sets in such a case are not necessarily automatically transitive.

We then look at the shortest possible *labels* that contain these code points and sequences:

|  |  |  |  |
| --- | --- | --- | --- |
| Initial label | Partitions | Variant labels | Comments |
| 0905 0902 | {0905 0902} {0905} {0902} | 0905 09020905 093A0905 0A020973 | original/index(not a valid label)(mixed script) |
| \*0905 093A | {0905 093A} {0905} {093A} | 0905 09020905 093A0905 0A02*0973* | indexoriginal/(not a valid label)(mixed script) |
| 0905 0A02 | {0905 0A02} {0905} {0A02} | 0905 09020905 093A0905 0A02*0973* | index(not a valid label)original / (mixed script) |
| 0973 | {0973} | 0905 0902*0905 093A**0905 0A02*0973 | index(not a valid label)(mixed script)original |

In the above, the first variant *label* set (for 0905 0902) allows four variant labels, based on the two ways the label can be partitioned into repertoire elements. When enumerating variants or generating index variants, all possible partitions are evaluated. The index variant is underlined (lowest in code point order).

If the variant label set were fully transitive, all four labels in it would have the same variant set containing the same four labels. However, the sets are not fully transitive as specified, and some variant labels are "inaccessible" from some of the other three labels in the set; they are shown in italics. Note that in some cases, valid labels cannot be reached. (This is the case, even if we were to exclude invalid labels as starting point – marked with asterisk).

Note, however, that in all cases, the index variant is the same and can be reached from all labels. Therefore, even though the variant *label* set is not transitive, all four labels still generate the same index and would be blocked from each other.

This only works because the one variant accessible to all happens to be the lowest in code point order.

Because no other variant sets in the LGR contain 0902, 0905, 093A or 0A02 either directly or as part of a sequence, there are no overlapping partitions possible, beyond the ones we have analyzed here.

***Option 3-1:*** Do nothing

The variant mappings as defined happen to give the correct result in index variant generation. Why change anything? The problem is that the way the two variant sets are defined hides their relatedness. Users (or mechanical verifications software) would have to generate the equivalent of the table above to find that the relation exists and that (by accident) it does not affect variant label generation.

***Option 3-2:*** Extend the *code point* variant set for the sequence (Set 2)

Add all mappings that result from the other partition of 0905 0902:

|  |  |
| --- | --- |
| Variant Set 2 | 0905 0902 🡨🡪 0973***0905 093A* 🡨🡪 0973*0905 0A02* 🡨🡪 09730905 0902 🡨🡪 *0905 093A*****0905 0902 🡨🡪 *0905 0A02******0905 093A* 🡨🡪 *0905 0A02*** |

The five added mappings are shown in bold. The set is transitive, but also contains all the mappings one would reach by enumerating all partitions and applying variant mappings to them.

## Issues with added variants

If Option 3-2 is implemented, each of the added mappings contains one of two sequences that is not valid under the Devanagari script LGR; these invalid sequences are shown in italics and color. The sequence 0905 093A is not valid, because it violates the code point context rule for 093A. The sequence 0905 0A02 is not valid, because it is mixed script. (Note however, that index variants are computed using the common LGR, and for that LGR, both scripts are in the repertoire. But, the sequence would also violate the code point context rule for 0A02, which is part of the common LGR).

Both sequences would have to be entered in the Devanagari LGR as "out-of-repertoire" otherwise they would become valid items in the Devanagari LGR, which is not desired. Adding these sequences as out-of-repertoire would cause a mismatch when merging LGRs: today, all out-of-repertoire variants in some LGR are also in-repertoire variants in another LGR. Therefore, after merging, no out-of-repertoire variants remain; the absence of such variants is a nice check that the LGR definitions are compatible.

This option seems a lot of effort to define things that can "never happen"; especially if the index variant happens to be computed correctly without these additions.

If there is a reliable way to verify that adding these additional mappings cannot change the result of index variant computation, it may be preferable to avoid these mappings. (This needs to be reviewed further).

Other situations may exist, where it is not possible to create the correct index variant without admitting an out-of-repertoire sequence. In that case, some or all of these mappings would have to be added, and a resolution found for handling the side effect of having such kinds of out-of-repertoire sequences. Such a situation might have arisen in the variant set analyzed here, for example, if the relative position of the Devanagari and Gurmukhi scripts in Unicode code space had been switched, which would have made the "mixed script" label the Index Variant.

# Full Analysis of Vowel plus Nukta Variants

In the following analysis, we assume that all mappings have been defined for blocked variants:

|  |  |
| --- | --- |
| Variant Set 1 | 0906 🡨 🡪 0906 093C |

We now treat both source and target as possible labels and look at their variants:

|  |  |  |  |
| --- | --- | --- | --- |
| **Label** | **Partitions** | **Variant labels** | **Comment** |
| 0906 | {0906} | 09060906 093C | original/index |
| 0906 093C | {0906 093C}{0906} {093C} | 09060906 093C0906 **093C** 093C | indexoriginal(not a valid label) |

*Code point* Variant Set 1 is symmetric and trivially transitive. However, both source and target contain some of the same code points in the same order.

The effect of this overlap between source and target of *code point* Variant Set 1 is to make 093C (Nukta) a variant of "nothing" whenever it follows vowel 0906. This means that any variant label set based on this variant definition will always have one label that can generate a variant one code point longer than can be generated from some other label in the same set.

0906 🡪 0906 093C

0906 093C 🡪 0906 **093C** 093C

…

As a result, it is not possible to create variant **label** sets that are transitive (where all members can be reached from any member).

In this example, it turns out that 0906 **093C** 093C and any labels with additional Nuktas in the sequence are not valid labels, due to a *code point* context rule on 093C. As discussed above, only *variant* context rules are processed during index variant computation.

**Option 4-1:** Do nothing

The variant label 0906 093C 093C is never enumerated and as it is not a valid label, no index variant for it needs to be computed. All other labels reach the same index variant: 0906. In principle, doing nothing is a valid option. The downside of it is that it requires this kind of analysis to prove that it works. Other scripts may have formally equivalent mappings that give different results.

**Option 4-2:** Add a *variant* context rule

We could add a context rule to Variant Set 3:

|  |  |  |
| --- | --- | --- |
| Variant Set 1 | 0906 🡨 🡪 0906 093C | not-when(followed-by-093C) |

The effect of this context rule is to make "undefined" the mapping when it is followed by 093C. (Since 093C is a Nukta, in the actual LGR such a rule would be named "followed-by-N".)

|  |  |  |  |
| --- | --- | --- | --- |
| **Initial label** | **Partitions** | **Variant labels** | **Comment** |
| 0906 | {0906} | 09060906 093C | original/ index |
| 0906 093C | {0906 093C}{0906} {093C} | 09060906 093C | indexoriginal |

Because of the added context rule, the partition {0906} {093C} can no longer generate a variant 0906 **093C** 093C. This makes the variant *label* set transitive.

## Accounting for cross-script variant for Nukta

Nukta (093C)also has a cross-script variant in 0A3C.

|  |  |
| --- | --- |
| Variant Set 4 | 093C 🡨 🡪 0A3C |

How does one account for this, given that Nukta is part of a code point variant sequence?

Formally applying this to the permutation would give:

|  |  |  |  |
| --- | --- | --- | --- |
| **Initial label** | **Partitions** | **Variant labels** | **Comment** |
| 0906 | {0906} | 09060906 093C | original/ index |
| 0906 093C | {0906 093C}{0906} {093C} | 09060906 093C0906 0A3C | indexoriginal |
| \*0906 0A3C | {0906} {0A3C} | *0906*0906 093C 0906 0A3C | index / (mixed script)original |

While both Variant Set 1 and 4 are formally transitive, the variant label set above is not, and, worse, the index variants are different. However, 0906 is not a valid code point context for 0A3C (even in the context of the common LGR). For possible whole-label variants like 0917 093C / 0A17 0A3C, we no longer have a contribution from Variant Set 1.

**Option 4.1-1:** Do nothing

Because the variant set 0906 🡨 🡪 0906 093C cannot occur in context that 0A3C can occur, we could simply do nothing. As explained in the background, it is not a requirement that invalid labels produce correct index variants.

The downside of doing nothing is that one requires analysis on this level to guarantee that the variant formalism works as intended.

**Option 4.1-2:** Add 0906 🡨 🡪 0906 0A3C to make variant label set formally transitive

This would become an out-of-repertoire variant in Devanagari. In Gurmukhi both source and target would be out-of-repertoire, which is irregular. Possibly not a good solution therefore.

# Full Analysis of Kashmiri Vowel 0974

In the following analysis, we assume that these mappings have been defined for blocked variants:

|  |  |
| --- | --- |
| Variant Set 1 | 0902 🡨🡪 093A0902 🡨🡪 0A02093A 🡨 🡪 0A02 |
| Variant Set 2 | 0906 0902 🡨🡪 0974 |
| Variant Set 3 | 0906 🡨 🡪 0906 093C : not-when(followed by 093C) |
| Variant Set 4 | 093C 🡨 🡪 0A3C |

*Code point* Variant Set 1 is fully transitive and the other ones are trivially transitive. Note the added context rule for 0906.

We then look at the shortest possible *labels* that contain these code points and sequences, but there is an interaction between Variant Set 2 and Variant Set 3 that forces us to consider certain code point sequences with 093C (Nukta):

|  |  |  |  |
| --- | --- | --- | --- |
| **Initial label** | **Partitions** | **Variant labels** | **Comments** |
| 0906 0902 | {0906 0902} {0906} {0902} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A02*0906 0A3C 09020906 0A3C 093A0906 0A3C 0A02*0906 0A020974 | original/index(not a valid label)(mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| \*0906 093A | {0906 093A} {0906} {093A} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A02*0906 0A3C 09020906 0A3C 093A0906 0A3C 0A02*0906 0A02*0974* | indexoriginal/(not a valid label)(mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| 0906 093C 0902 | {0906 093C 0902}{0906 093C} {0902}{0906} {093C} {0902} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| 0906 093C 093A | {0906 093C 093A} {0906 093C} {093A}{0906} {093C} {093A} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original / (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| \*0906 093C 0A02 | {0906 093C 0A02} {0906 093C} {0A02}{0906} {093C} {0A02} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original / (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| \*0906 0A02 | {0906 0A02} {0906} {0A02} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A02*0906 0A3C 09020906 0A3C 093A0906 0A3C 0A02*0906 0A02*0974* | index(not a valid label)(mixed script)(mixed script)(mixed script)(mixed script)original / (mixed script) |
| 0974 | {0974} | 0906 0902*0906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974 | index(not a valid label)(mixed script)(mixed script)(mixed script)(mixed script)(mixed script)original |
| \*0906 0A3C 0902 | {0906 0A3C 0902}{0906 0A3C} {0902}{0906} {0A3C} {0902} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| \*0906 0A3C 093A | {0906 0A3C 093A} {0906 0A3C} {093A}{0906} {0A3C} {093A} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original / (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |
| \*0906 A93C 0A02 | {0906 0A3C 0A02} {0906 0A3C} {0A02}{0906} {0A3C} {0A02} | 0906 09020906 093A0906 093C 09020906 093C 093A0906 093C 0A020906 0A3C 09020906 0A3C 093A0906 0A3C 0A020906 0A02*0974* | index(not a valid label)original / (mixed script)(mixed script)(mixed script)(mixed script)(mixed script) |

We proceed as before: In the above, the first variant *label* set (for 0906 0902) allows six variant labels, based on the two ways the label can be partitioned into repertoire elements. When enumerating variants or generating index variants, all possible partitions are evaluated. The index variant is underlined (lowest in code point order).

If the variant label set were fully transitive, all six labels in it would have the same variant set containing the same six labels. However, the sets are not fully transitive as specified, and some variant labels are "inaccessible" from some of the other three labels in the set; they are shown in italics and color. Note that in some cases, valid labels cannot be reached. Also, some labels can be partitioned three ways.

The interaction between Variant Set 2 and Variant Set 3 caused us to include certain labels of length 3. Because of the context rule, these in turn cannot create variants of length 4, which otherwise would have been "inaccessible" variants to any of the shorter labels. Variant Set 4 increases the number of permutations to 10, although many are not valid labels. All ten variant labels produce the same index variant meaning it can be reached from all labels. Therefore, even though the variant label set is not transitive, all six labels still generate the same index and would be blocked from each other.

As before, this only works because the one variant accessible to all happens to be the lowest in code point order.

**Option 5-1:** do nothing

This relies on the fact that the numbers just work out to make the generated index variant the same and on the other fact that because of code point context rules none of the labels with more than one Nukta (093C) are valid labels, so we would not have to create index variants for them.

It is however, not satisfying to rely on these two facts when they are not readily apparent from the specification of the LGR, except by working out full examples like the one here.

**Option 5-2:** Extend the *code point* variant set for the sequence (Set 2)

As before in the example for 0973. The same caveats and need for further study apply.

Add all mappings that result from the other partition of 0906 0902:

|  |  |
| --- | --- |
| Variant Set 2 | 0906 0902 🡨🡪 0974***0906 093A* 🡨🡪 0974*0906 0A02* 🡨🡪 0974****0906 093C 0902 🡨🡪 0974*0906 093C 093A* 🡨🡪 0974*0906 093C 0A02* 🡨🡪 0974****0906 0902 🡨🡪 *0906 093A*0906 0902 🡨🡪 *0906 0A02**0906 093A* 🡨🡪 *0906 0A02*** |

The eight added mappings are shown in bold. The set is transitive, but also contains all the mappings one would reach by enumerating all partitions and applying variant mappings to them.

**Option 5-3:** also extend to account for Variant Set 4

This option does not cover any variant labels that would be valid. (See discussion above for Nukta).

## Issues with added variants

As before, all but one of the mappings contains one of two sequences that is not valid under the Devanagari script LGR; these invalid sequences are shown in italics and color. The sequence 0906 093A is not valid, because it violates the code point context rule for 093A. The sequence 0906 0A02 is not valid, because it is mixed script. (Note however, that index variants are computed using the common LGR, and for that LGR, both scripts are in the repertoire. But, any mixed script sequence would also violate the code point context rule for 0A02, which is part of the common LGR).

The sequence 0906 093C 0902 is a valid sequence; this makes the current example different from Option 3-2 for U+0973.

The same caveats apply as for Option 3-2 for U+0973. Again, the numeric code point values are such that the Index Variant is a valid label, and all of the possible additions are labels that should never happen. This is not something that we can currently verify automatically; but it remains unattractive to add lots of "impossible" mappings for formal completeness when there cannot be a difference in practice.

 (This needs further review).