Supporting Sequences in an LGR

DATE: 2019-09-24

# Overview

This document describes a number of issues encountered in LGRs that support sequences. The examples presented here are given in a simplified notation that focuses on the logical aspects, not the full xml syntax defined in RFC 7940. This document should be read in the context of RFC 7940 and RFC 8228. (The latter uses a similar simplified notation for presenting examples and shows how to relate that notation to the XML syntax specified in RFC 7940).

# Sequences as elements of the Repertoire

RFC 7940 allows both individual code points as well as sequences as members of the LGR’s repertoire.

Assume an LGR that defines these code points:

A  
B

if the LGR also defines the sequence

AB

then that sequence is ***redundant.*** When a label “AB” is defined it could either be {A}{B} or {AB}, where the { } indicate a possible *partition* of the label into repertoire elements. Removing the definition of such a sequence has no effect on the permissible labels under the LGR.

If the LGR were to define a sequence

ABC

that sequence is not redundant, but instead allows code point C to occur in a label if and only if it is preceded by AB. The label “ABC” has only one partition {ABC}. Removing the definition of this latter sequence would disallow any labels containing C.

Such sequences provide an ***enumerated context*** for code point C and make it ***indirectly available*** for use in labels. Generally, enumerated sequences are to be preferred for cases where some code point is used in very few combinations with other code points, or when the sequence represents something that users treat as a unit.

However, logically, the same effect can in principle be achieved with a detailed context rule (see below).

# Sequences and Variants

A redundant sequence that is the ***target of a variant mapping*** that is different from the variants for its elements would not be redundant.

A 🡪 D  
B  
D 🡪 A  
E 🡪 AB  
AB 🡪 E

In the above example, a label “AB” can have the following variants

{A}{B} has variants AB (original) and DB

while the other partition

{AB} has variant AB (original) and E

Removing sequence AB from the LGR would change the variants that exist for label “AB”, and it is therefore no longer redundant.

# Sequences and Context Rules

Code points and sequences can have ***context rules***. Required contexts are shown with a “when(context)” following a colon. (Prohibited contexts are also possible and would use “not-when”). Assume the following LGR containing two code points C and M, where M has a context rule:

C  
M: when (follows-C)

In an LGR as specified above, labels “MC” or “MM” would not be possible, but “CC” or “CM” would be.

A sequence

CM

would be redundant, because a label “CM” is already allowed. However, a sequence

CMM

could be defined to ***override*** the contest restriction on the second M. Such a sequence is not redundant, because if it were removed from the LGR, the label “CMM” would no longer be valid. It is important to understand here, that inside a sequence, the context rules for code points like M are not evaluated.

For the partitions of the label “CMM”

{C}{M}{M} — this partition does not lead to a valid label, because the second {M} does not satisfy the context.

{CMM} — this partition leads to a valid label, because it matches sequence CMM and there is no context rule applied to either of the two M in that sequence.

Note that the label “CMMM” remains invalid, as the context for the last M is still “when(follows-C)”. In evaluating that context, it does not matter that the preceding code point is part of a sequence, so for example a sequence

CMMC

would be a valid part of a label “CMMCM”.

Note that defining a variant for a sequence

MC 🡪 XX  
XX 🡪 MC

would result in a labels “CMC” and “CX” to be variants or each other. With the context rules defined for C and M above, such a label would be valid. Unfortunately, defining a sequence MC has the *side effect* of also allowing labels such as “MC”, because the sequence would override the context rule on M.

Modifying the LGR to give the sequence a *matching context rule* resolves this issue.

MC : when(follows-C) 🡪 X

With this context rule, the M in the sequence MC can now occur only in the same contexts as any singleton M, therefore, the sequence no longer overrides a context rule.

# Sequences and Variant Contexts

Variant contexts are described in RFC 7940 and discussed at length in RFC 8228.

In the preceding example, the code point X has no context rule, which would allow labels like CMXX which would then have the variant CMMC, which is an invalid label because of the context rule on MC.

For some LGRs a sequence like XX is itself needed only as a target for the variant (for example, because X is specified on its own). In that case, the sequence XX can be given a matching context rule. Here is the “full” LGR for this scenario:

C  
M : when(follows-C)   
MC : when(follows-C) 🡪 XX  
X  
XX : when(follows-C) 🡪 MC

In other cases, a variant for something like MC may to a sequence (or singleton) that is not redundant and cannot be restricted to the same context as MC (or it has a different context restriction that partially overlaps that for MC). In all of these cases, we would add a context rule to a the variant itself.

Assuming Y is a singleton that should not have a context restriction, the LGR would look like:

C  
M : when(follows-C)   
MC : when(follows-C) 🡪 : when(follows-C) Y  
Y 🡪 : when(follows-C) MC

Note the way the context rules are applied separately to the sequence and the variant mapping.

A context rule on a repertoire element (code point or sequence) that is not satisfied leads to an invalid label. A context rule on variant mapping that is not satisfied is treated as it the mapping is not defined in such condition.

Noted that context rules on variants should always match in both directions, in other words A🡪B and B🡪A should always have the same variant context (or none).

The variant context should be the more constrained (or the intersection of) the code point contexts for A and B. If the code point context for A and B is the same, a variant context is not necessary.

Please see RFC 8228 and LGR-3 overview for discussion of additional edge cases concerning variants and sequences (such as overlapped and effective null variants).