L2/19-262

Properties that Apply to Sequences

DATE: 2019-07-12
SUBMITTED BY: Asmus Freytag, PhD, Technical VP Emeritus, The Unicode Consortium

1 Overview

There are a number of places where the Unicode Standard contains what are effectively properties for sequences, but their documentation and availability in the UCD is uneven.

This proposal provides some background on a current project that successfully uses sequences on par with singleton characters as part of defining and evaluating non-trivial policies for complex-script identifiers, including by defining mappings and other properties to them.

The remaining sections mainly list cases where Unicode defines sequences that de-facto have several types of properties (beyond those already discussed in other proposals on this subject).

They also discuss one property that currently exists for code points in a security related context, but, as our work described in the background section has surfaced, should be extended to sequences so as to not be crucially incomplete.

The document concludes with a discussion of how to incrementally get there.

2 Background

The following discussion describes the author's experience with developing a system that processes sequences that have mapping and other properties on nearly equal footing with ordinary characters.

For about the last decade, ICANN has been engaged in a process that will culminate in the development of Label Generation Rules for IDN TLDs in the DNS Root Zone. (See [RZ-LGR-3]). As the name implies, these go beyond simply applying restrictions to individual code points, but they apply potentially to the whole label, in other words, a string or sequence of code points.

In the process, Kim Davies of ICANN and I developed RFC 7940 which allows for a machine readable (XML-based) description of these rules. Key features of this RFC include being able to define both individual code points as well as sequences as elements of the repertoire. Sequences (with one exception) are fully equivalent to singleton code points: they can be source and target of variant mappings, or have context rules defined for them. Context rules, as well as whole-label rules are expressed as nested XML elements that can be translated directly into regular expressions.

The syntax for context and whole-label rules allows character classes based on explicitly listed sets, based on Unicode properties, or based on attributes ("tags") applied to the character elements in the

repertoire. The last is effectively a mechanism for local definitions of properties, and is one of the most widely used features of the RZ-LGR.

At the time, we did not allow sets of sequences nor local properties for them; that decision was based on the absence of a direct equivalent in the common subset of widely available regular expression engines. For most purposes, a ready work-around exists in form of a choice expression combining sequences with a character class. If “ss” is a sequence, and a and b members of a character class, such an expression would be ([ab]|ss). As our syntax allows named subexpressions (as well as named character classes) the impact of this workaround on readability of the resulting expressions is contained.[1]

However, there are other areas where sequences participate more fully. It may be worth detailing our experiences with this mixed processing model, because it shows the applicability of properties (including mapping properties) for sequences outside of a pure regular expression context. This processing model requires careful attention to some subtleties.

In a domain label, especially for complex scripts, not all code point sequences are possible. Restrictions are generally expressed as context rules for individual repertoire elements; these are effectively anchored regular expressions that adjacent code points either satisfy or not. (In addition, whole label rules may also be defined; these would be equivalent to regular expressions over the whole label).

When sequences are defined, they override the context rules for their constituent code points, and replace those by context rules that apply to the code points surrounding the sequence. For example, if a mark m can only directly follow a certain base character c, a sequence of type c-m-m would override this restriction. The reverse is not true: if a sequence were to be defined that uses a different base character b, that is b-m, it would not invalidate a c-m sequence that is permissible under the context rules for c and m.

Another use of sequences can be in enumerating a subset of combining sequences. This is done by not adding the combining mark to the repertoire by itself, so it can only occur as part of a sequence.

One of the main functions of Label Generation Rules is to define whether two code points or sequences are variants of each other. A variant mapping has a source and target (which may be the same, for reflexive variants) and a type value, which ultimately is used to resolve the disposition of the variant label. (A variant label thus “remembers” which mappings were used to create it).

In evaluating variants, all partitions of a label into code points and sequences need to be investigated. (For context rules, the analysis can stop once some partition leads to a valid label). Each partition is then

[1] Context rules reference local and other properties via named or implicit character classes that are constructed from all code points sharing the property (or a combination of properties). Named classes and named subexpressions are syntactically similar, so that having to create and reference a subexpression is not too onerous. However, one loses the self-documenting feature of associating a local property value directly with a sequence in the listing of the repertoire. If we were redesigning this, that might be something we would fix.

mapped to a variant label, by applying a permutation of all variant mappings for the specific sequences and code points that make up that partition of the label.

When evaluating a regular expression, the partitioning of the input label is built into the process, but for other tasks, like variant mappings, it has to be taken into account explicitly.

Introducing sequences into variant mappings adds some edge cases, for more details see RFC 8228 or [RZ-LGR-3].

Finally, it should be noted that neither in developing the RZ-LGR nor RFC7940 did we consider limiting the sequences to those that represent complete combining sequences. In fact, in several scripts we cover, there are interesting subsequences of full combining sequences that have characteristics (properties, one might say) that make it preferable to define operations and constraints based on subsequences.

This is really no different from properties on individual combining characters. While in some cases a combining mark can be treated as if inheriting the base characters properties, so that a European letter+diacritic sequence can be conceptualized as simply an extended single letter, this approach is far from natural for most complex scripts. (Just compare the Indic Syllable Categories which likewise assigns properties to combining characters).

The purpose of this rather deep background digression was to show that real-world projects exist where character sequences are full members and are largely treated as on the same footing as singleton characters.

Incidentally, in the tool that provides a human-readable HTML presentation of the LGR's XML, we utilize one place where Unicode already provides a machine-readable property for sequences: named sequences. If an LGR defines a sequence that corresponds to a Unicode Named Sequence, it is automatically annotated with the correct name.

3 Some Properties That Apply to Sequences

3.1 Named Sequences

Named Sequences are exposed in the UCD. They assign the "name" property to a subset of sequences.

3.2 Standard Variation Sequences

Standard Variation Sequences are exposed in the UCD. They implicitly assign an equivalent (non-varied) code point as well as a glyph property to the sequence. There is finally an informal property defining the type of variation.

3.3 Do not Use

Chapters 11 and 12 (and several others) of the Unicode Standard contain tables of sequences that are effectively deprecated ("do not use") while their constituent characters are not. This is a de-facto property definition, presented in tabular form in the text, yet not exposed in the UCD.

[While the actual implementation of RZ-LGR-3 primarily uses context rules based on character class to express these constraints, it would have been a tremendous benefit to have the information from those tables available in machine-readable form – if for no other purpose than to verify that the rule systems were effective in preventing all “do not use” sequences.]

3.4 Equivalent Sequence

The same chapters list equivalent sequences for both do not use sequences as well as some others. While in many cases the equivalence may be between a single code point and a sequence, the property is better conceived as generalized sequence to sequence mapping (where singletons are sequences of length 1). This, again, is a de-facto property definition, yet not exposed in the UCD.

3.5 Intentionally Identical

The UCD provides an “intentionally identical” property for characters as part of the security data. There are a number of cases where this could (and we argue should) be extended to cover sequences that are equally identical in appearance.

Examples include

1004 103A MYANMAR LETTER NGA + MYANMAR SIGN ASAT
105A 103A MYANMAR LETTER MON NGA + MYANMAR SIGN ASAT

178A 17D2 ◌្ KHMER CONSONANT SIGN COENG DA
178F 17D2 ◌្ KHMER CONSONANT SIGN COENG TA

The former example is formally a complete combining sequence while the latter is not (it can follow any consonant in that script). While it would be possible to restate the latter case by enumerating all full combining sequences it would just obfuscate the reality that it is the subjoined consonants that are identical in appearance (but not in phonetic value). The last two sequences above are named sequences (with the sequence name shown).

Another example of identical sequences would be the legacy TAMIL SHRII vs. the legacy equivalent TAMIL “SRII”

0BB6 0BCD 0BB0 0BC0 ஶ்ரீ TAMIL SYLLABLE SHRII
0BB8 0BCD 0BB0 0BC0 ஸ்ரீ TAMIL LETTER SA + TAMIL SIGN VIRAMA + TAMIL LETTER RA + TAMIL VOWEL SIGN II

Identical sequences intersect with equivalent sequences to a degree. The former should include only cases where both sequences are in use and neither subject to normalization nor to effective deprecation (“do not use”) that would disqualify one of them from a security related context such as identifiers. The latter more generally show alternatives whether strongly, or simply preferred or without preference.

These examples all surfaced as part of the RZ-LGR-3 development, where they were handled by explicitly assigning a variant relation between these pairs. Zone administrators and bodies developing rules for

other zones of the internet are looking to the Unicode Consortium as an authoritative source for information like this. It is therefore incumbent on the UTC to find a way to extend the property model beyond singleton code points and cover highly security relevant properties like these for sequences.

4 Incremental Deployment

In a way, sequences are different from code points: they are open ended and do not form a finite set. This raises questions of completeness for property assignments. However, this is not that different from the cases that are already handled as de-facto properties. Unicode has named only a very small subset of sequences to date, but leaves open the possibility of future additions, as conditions warrant. In that example, stability is guaranteed for properties (names) already assigned, but not for the status of "unnamed" sequence.

The projects cited as background for the current proposal are for project-specific reasons purposefully limited to letters and, in particular, to letters for modern scripts (as defined in [MSR-4]). It can be argued that collecting information for sequences could likewise begin with this and other high-use subsets and then incrementally increase to involve other scripts.

Because of the open-ended nature of sequences, the "missing" property should probably never be considered stable: the general presumption would be that additional sequences could be assigned the given property in the future, irrespective of whether the property values themselves are considered stable or adjustable.

In effect, this would simply formalize and rationalize the evolving information that the Unicode Standard already supplies for sequences.

This document does not propose the individual updates that would be necessary to the Unicode Character Property Model or the definitions, but the author would be happy to contribute to any editorial work necessary.

5 References

[MSR-4] Integration Panel, "Maximal Starting Repertoire — MSR-4 Overview and Rationale", 7 February 2019 https://www.icann.org/en/system/files/files/msr-4-overview-25jan19-en.pdf [PDF, 0.8 MB]

[RFC7940] K. Davies, A. Freytag, Representing Label Generation Rulesets using XML, https://datatracker.ietf.org/doc/rfc7940/

[RFC8228] A. Freytag, "Guidance on Designing Label Generation Rulesets (LGRs) Supporting Variant Labels", RFC 8228, August 2017, http://www.rfc-editor.org/info/rfc8228

[RZ-LGR-3] Integration Panel, “Integration Panel: Root Zone Label Generation Rules — LGR-3”, 10 July 2019, https://www.icann.org/sites/default/files/lgr/lgr-3-overview-10jul19-en.pdf