Integration Panel:

Whole Label Evaluation (WLE) Rules

REVISION 21 – 2017-09-15

Background

What are Whole Label Evaluation (WLE) rules

Whole label evaluation rules determine the validity of a label based on whether its code points appear in permissible contexts. They are called "whole label" because the context may, in some cases, involve the whole label. However, in many cases of practical relevance, rules based on the immediate context (adjacent code points or label boundaries) are sufficient to decide whether the label should be valid. Because any label that consists entirely of code points in the repertoire is normally valid, the effect of a WLE rule is to further restrict the set of valid labels. For example, a WLE rule may not permit labels that start (or end) with certain code points. WLE rules can be formulated to apply to specific code points or classes of similar code points. They can be used to evaluate an entire label, or just the context around a specific instance of a particular code point at a given position in the label.

Why are WLE rules needed

Many kinds of code points can occur in any position and in any combination in a label without causing any problems. Labels are intended to be mnemonics and there is no requirement that they form actual words in any language. However, certain code points can cause issues in processing and presentation if they appear out of their intended context. Labels that cannot be processed or rendered reliably present a risk of confusion or a risk to DNS security. WLE rules exist to prevent such labels.

In contrast, WLE rules are not intended to enforce "spelling rules" for any particular orthography, such as requiring that the letter "q" always be followed by the letter "u".

When to consider WLE rules

WLE rules may be considered whenever some sequences of code points should be generally prohibited within a script, and when it is not possible to prevent such sequences by removing some or all of their constituent code points altogether from the repertoire. Such prohibited sequences are particularly important wherever the display of the code points outside these contexts becomes unpredictable for a given script or gives rise to a security risk. A typical example of this would be combining marks, code points that graphically combine with the preceding code point. While Unicode does not restrict their application, most rendering engines are only prepared to deal with combinations actually required or allowed in a given language or script.

For many scripts, WLEs may not be needed. Their benefit may not be compelling and their complexity cost may be high. If a label merely 'looks bad' when ill-formed, that may not be a reason to filter it. The

same applies to sequences of code points that 'can't happen' in a given writing system, but that merely look incorrect to the native reader, while not posing issues to a rendering system, or for security.

Simple requirements such as pair context can often be addressed by declaring the pair a member of the repertoire as itself, while making sure that at least one code point in each sequence is not also part of the repertoire. To do so enforces the requirement that the code point may only occur as part of that specific sequence.

For example, if a certain combining mark occurs with only one or a few base code points, the repertoire could list the permissible sequences, but not the combining mark itself. This makes sure that labels containing permissible sequences are valid, but prevents arbitrary application of the combining mark. This simple technique avoids the need to create a WLE rule. If needed, the approach can be extended to longer sequences as well. Rule based approaches are more suitable where marks can occur in many contexts.

Complex scripts

In general, WLE rules are more appropriate for code points in "complex scripts" as opposed to "simple" scripts. A simple script is one where each code point forms an isolated unit, and code points can generally appear in any order (combining diacritics would be an exception). Complex scripts are those that have complex layout behavior, and only a subset of all possible code point sequences would ever expected to occur. The unit of writing is generally a cluster of code points, or even a full syllable. Inside a syllable, the constituent code points corresponding to consonants, vowels, tone marks etc. do not occur in arbitrary positions, but reflect a syllable structure specific to the script.

Appropriate WLE rules

WLE rules can be used to limit the context in which certain code points or marks may appear, so that they fall in the range expected (and supported) by typical rendering engines.

Examples of WLE rules that might be appropriate include:

- limiting combining diacritics to those combinations that are required (unless handled by adding the sequence, but not the diacritic, to the repertoire)
- disallowing vowel marks where they can't be rendered, such as at the start or following other vowel marks
- disallowing tone marks where they aren't expected, such as following another tone mark, or not following a vowel
- disallowing marks like halants, that are intended to act on consonants, from appearing after vowels
- disallowing multiple combining marks (diacritics) applied to the same base letter, for writing systems where such use is not required

What contexts can WLE rules describe?

WLE rules effectively implement a subset of the functionality of common regular expressions, but with a few extensions. They support these common features:

- literals,
- greedy repetition (0 or more, 1 or more, minimal, maximal repetition) with yielding,
- character classes (enumerated or based on Unicode properties),
- set operations on character classes,
- grouping
- alternatives (or)
- positional operators (start / end of label)

They also support the following extensions:

- common subexpressions
- anchored evaluation (parameterized)

The former perform like macros, the latter allows a rule to be evaluated at a particular location in the label and substituting the actual code point at that location as if it was a literal in the regex. Anchored evaluation is used to implement context rules for a given code point, while all the other features are used for ordinary WLE rules that evaluate a whole-label.

How are WLE Rules expressed in XML?

WLE rules can be expressed in the formal XML syntax for Label Generation Rules [RFC7940] in several ways. The simplest way is a context rule associated with a specific code point. WLE rules can define constraints in two ways:

- Required contexts – positive assertions that the code point must appear in a certain location or together with specific code points; or
- Prohibited contexts – negative assertions that the code point must *not* appear in a certain location or together with specific code points.

In both cases, the intent of the WLE rule is to specify which labels are not valid – even though they contain only valid code points – due to not meeting the constraints on context for these code points in the given label. Whether a context is defined as required or as prohibited is a choice that would be based on the nature of the restriction, and sometimes on whether it makes it easier to express the restriction in [RFC7940] as one or the other.

In [RFC7940], such contexts are described as named <rule> elements, which are associated with a given code point via a “when” or “not-when” attribute that refers to the context by name.

```
<char cp="0000" when="some-required-context" />
<char cp="0001" not-when="some-prohibited-context" />
```

The context rules may contain an <anchor> element as placeholder for the code point for which the context is to be evaluated. When an <anchor> is used, any context before or after it in the same <rule> must be enclosed in a <look-behind> or <look-ahead> element.

```
<rule name="some-required-context">
```

```
  <look-behind>
    <char cp="0002" />
  </look-behind>
  <anchor />
</rule>
```

The preceding rule defines a sample context where the code point in question follows code point U+0002. Whether this is a required or prohibited context is defined by how it is used. If used with a “when” attribute it acts a *required* context, if used with the “not-when” attribute it acts as a *prohibited* context. Each label is scanned for the contexts associated with any “when” or “not-when” attributes for each of its code points. If a code point is found in a prohibited context, or conversely, if the code point is not found in a required context, then the code point at that position in the label is deemed invalid, invalidating the label.

Some rules require the context of the full label. They are expressed in [RFC7940] as rules that do not contain an <anchor> element. In some cases, they may not even be associated with specific code points. Instead, they would be associated with one or more <action> elements by using a “match” or “not-match” attribute as in the following examples:

```
<action disp="invalid" match="ill-formed-label" />
<action disp="invalid" not-match="well-formed-label" />
```

Here “ill-formed-label” and “well-formed-label” are assumed to be the names of <rule> elements matched by certain ill-formed or well-formed labels respectively.

Each action is evaluated at most once per label, and if a label matches a rule describing an ill-formed label (or in the second case, if it doesn’t match the description of a well-formed label) the action causes a final disposition to be assigned to the label, as specified in the action’s “disp” attribute. For example, when an <action> element has the disposition of “invalid” the rule and type of match attribute on the action are chosen in such a way that the action is only triggered if a label is invalid. That label is then given the disposition “invalid”.

The choice of whether a rule is constructed as positive (matches valid labels) or negative (matches invalid labels) does not affect the result as long as the correct match attribute is chosen. In many cases, one or the other expression, though equivalent, is easier to understand and verify than the alternative, making it the preferred choice.

More details on how to specify WLE rules in XML can be found in [RFC7940], [Regex-XML] and in the section on Examples of WLE rules below.

Examples of WLE rules

This section gives a number of examples of different types of WLE rules. The last one of these examples covers a strategy that can be used to reduce the complexity of certain rules, without necessarily compromising their benefits.

Context rules in IDNA2008

IDNA2008 contains a number of context rules, which cover a wide range of scenarios for WLE rules. In the IDNA2008 specification they are given in pseudocode. “IDNA 2008 Rules in XML” [IDNA-XML] shows how they can be translated into the XML format defined in [RFC7940].

Example 1, Restricting a combining mark to specific sequences

Code Point U+0331 COMBINING MACRON BELOW is used in various pre-composed letters (in the Unicode character names for these letters it is typically called LINE BELOW but it is the same diacritical mark). Normally, because a U-label in IDNA 2008 is in normalization form NFC, code points that can only be part of already composed sequences are not needed, because the sequences are normalized away in favor of the precomposed forms.

However, the African Reference Alphabet[1] defines letters that would require additional sequences (such as c, q, s, and x) with a line below and which are not already encoded. It might be desirable to allow those, but not any other sequences. A WLE rule could be defined to restrict the combining macron below to occur only after the letters c, q, s, and x.

Such a rule would require the definition of a class, or set of code points, (here called cqsx after the letters it contains) that includes all the supported base characters for the combining mark.

```
<class name="cqsx">
  <char cp="0063" />
  <char cp="0071" />
  <char cp="0073" />
  <char cp="0078" />
</class>
```

This definition is referenced in a rule that requires the code point for which the rule is evaluated (the “anchor”) to be in the context defined by the rule. Here, the code point must follow the class “cqsx”.

```
<rule name="follows-cqsx">
  <look-behind>
    <class by-ref="cqsx" />
  </look-behind>
  <anchor />
</rule>
```

This rule would be invoked with a `when="follows-cqsx"` attribute for the code point U+0331:

```
<char cp="0331" when="follows-cqsx" />
```

When everything is put together, the effect of the XML fragments in this example is that the only letters that can occur with a line below are any of the precomposed letters as well as the combinations `<0063 0331>, <0071 0331>, <0073 0331> and <0078 0331>.`

[1] http://en.wikipedia.org/wiki/African_reference_alphabet

Alternatively, these sequences, but not the combining mark itself (0331), could have been directly added to the repertoire. This is a simpler and therefore preferable method whenever the permissible code point sequences are easily enumerated. For example:

```
<char cp="063 0331" comment="c with macron below" />
<char cp="071 0331" comment="q with macron below" />
<char cp="073 0331" comment="s with macron below" />
<char cp="078 0331" comment="x with macron below" />
```

A note on naming

It is convenient to name rules for left-hand contexts "following-XXXX" where "XXXX" is chosen to be as descriptive of the details of the context. For example, "following-cqsx" can be readily understood when seen where it is applied

```
<char cp="0331" when="follows-cqsx" />
```

whereas a name like "context-for-0331" can only be understood by reference to the actual rule itself. For right-hand contexts, the convention would be "precedes-XXXX". Because simple contexts are always preferable over complex ones, the majority of rule names would follow one of these two patterns.

Example 2, Intersyllabic tsheg

The Tibetan *intersyllabic tsheg* (U+0F0B) is a code point that is exceptionally PVALID in IDNA2008. Unicode classifies the code point as a punctuation mark, but the *tsheg* is required between all syllables of all polysyllabic words in Tibetan. While leaving this code point out of a zone repertoire would make it impossible to use readable polysyllabic words as mnemonic labels in Tibetan, it is probably a good idea to prevent extraneous *tshegs* in a label. That would require a WLE rule that prevents a *tsheg* from starting or ending a label, or from being adjacent to another *tsheg*.

This requirement is captured, for example, by the following regular expression that defines the three prohibited contexts, where T stands for the *tsheg*.

(^T|TT|T$)

Here ^ and $ stand for the start and end of the label. () and | indicate that there are three alternative contexts that are prohibited. Any label *matching* that regular expression would not be a valid label. [Regex-XML] provides a listing of elements in the XML format and their equivalent regular expression operators. Using this, we can translate the regular expression into the corresponding XML format.

```
<rule name="prohibited-for-intersyllabic-tsheg">
  <choice>
    <rule>    <!-- corresponds to ^T -->
      <look-behind>
        <start />
      </look-behind>
      <anchor />
    </rule>
    <rule>    <!-- corresponds to T$ -->
      <anchor />
      <look-ahead>
        <end />
      </look-ahead>
    </rule>
    <rule>    <!-- corresponds to TT -->
      <anchor />
      <look-ahead>
        <char cp="0F0B" />
      </look-ahead>
    </rule>
  </choice>
</rule>
```

In the XML format, the context is expressed as a single named rule (prohibited-for-intersyllabic-tsheg). Because it is essentially a compound of three alternatives, that rule in turn consists of a single <choice> element with three subordinate rules describing the three alternative terms of the regular expression above.

Each subordinate rule defines one of the ineligible contexts for an *intersyllabic tsheg*, with the special elements <start> and <end> referring to the beginning and end of the label, just like ^ and $ in a regular expression. Inside each subordinate rule, all elements are matched in sequence, while in the <choice> each subordinate rule is matched in order until there is a match for the entire context or there are no more subordinate rules left.

Note that with one exception, none of these subordinate rules references the *intersyllabic tsheg* (0F0B) directly. Instead they each contain an instance of an <anchor> element, which is a stand-in for the code point for which the rule is being evaluated.

To invoke the rule every time a label contains an *intersyllabic tsheg*, we attach it to the entry for code point U+0F0B in the <data> section of the XML file using a “not-when” attribute:

```
<char cp="0F0B" not-when="prohibited-for-intersyllabic-tsheg" />
```

Doing so attribute marks the *intersyllabic tsheg* as invalid whenever it is placed in a context that matches the rule referenced by the “not-when” attribute. When evaluating the rule, each <anchor> element is replaced by a *intersyllabic tsheg* at the given location. The rule would be evaluated repeatedly, once for every tsheg in the label.

There are indications that in actual texts the *tshegs* may occur at the end of words. If it is desirable to support this, the WLE rule for this code point would have to be revised accordingly.

Example 3, Syllables in the Thaana script

The Thaana script is written in syllables, but the encoding form chosen is that of an alphabet. As a result, all consonants, with one exception, must be followed by a vowel sign. Also, each syllable only has one vowel sign. Because Thaana vowels are encoded as combining marks, applying multiple vowels to a consonant not only creates forms that native readers will not understand, but could also lead to the marks being undetectably overprinted on top of each other, which would be a security risk. These restrictions being fundamental to the script itself, they might make a good candidate for a WLE rule.

There are several alternative, yet equivalent ways that these restrictions can be expressed in the XML format.

Alternative 1— Associate Code Points with Tags

The following defines the entire repertoire of the Thaana script, as used for IDNs, and associates each code point with a tag, indentifying code point as consonant or vowel. All code points, except U+0782, are also associated with one of two required contexts by using the “when” attribute.

```
<data>
 <range first-cp="0780" last-cp="0781" tag="consonant" when="precedes-vowel" />
 <char cp="0782" tag="consonant" /> <!-- no when rule -->
 <range first-cp="0783" last-cp="07A5" tag="consonant" when="precedes-vowel" />
 <range first-cp="07A6" last-cp="07B0" tag="vowel" when="follows-consonant" />
 <char cp="07B1" tag="consonant" when="precedes-vowel" />
</data>
```

The contexts define whether a code point must follow a consonant (true for vowels) or must precede a vowel (true for all but one consonant). Here are the XML fragments defining these contexts. Note how they refer to the classification of the code point by the tag values.

```
<rules>
  <rule name="follows-consonant">
    <look-behind>
      <class from-tag="consonant" /> <!-- any code point with tag="consonant" />
    </look-behind>
    <anchor />
  </rule>

  <rule name="precedes-vowel">
    <anchor />
    <look-ahead>
      <class from-tag="vowel" />  <!-- any code point with tag="vowel" />
    </look-ahead>
  </rule>
  ...
</rules>
```

Note how this set of contexts is really equivalent to requiring that every syllable be well-formed, thus essentially requiring each label to be sequence of well formed syllables. However, these rules do not introduce the concept of a syllable, nor do they necessarily evaluate the context of the entire label, resulting in a considerable reduction in complexity. We will use the same technique in a later example where the rules for syllable formation are more complex. But first, we show some alternative formulations of WLE rules that achieve the same results.

Alternative 2—Use Explicit Character Classes

This alternative formulation avoids the use of "tag" values in favor of defining explicit character classes.

```
<!-- same repertoire as before, except not using "tag" -->
<data>
 <range first-cp="0780" last-cp="0781" when="precedes-vowel" />
 <char cp="0782" /> <!-- no when rule -->
 <range first-cp="0783" last-cp="07A5" when="precedes-vowel" />
 <range first-cp="07A6" last-cp="07B0" when="follows-consonant" />
 <char cp="07B1" when="precedes-vowel" />
</data>

<rules>
  <!-- explicit definition of sets -->
  <class name="consonant" comment="set of consonants">
    <range first-cp="0780" last-cp="07A5" />
    <char cp="07B1" />
  </class>

  <class name="vowel" comment="set of vowels">
    <range first-cp="07A6" last-cp="07B0" />
  </class>
  <!-- same context rules as before, except using "by-ref" -->
  <rule name="follows-consonant">
    <look-behind>
      <class by-ref="consonant" />
    </look-behind>
    <anchor />
  </rule>

  <rule name="precedes-vowel">
    <anchor />
    <look-ahead>
      <class by-ref="vowel" />
    </look-ahead>
  </rule>
  ...

</rules>
```

The downside of explicitly declared classes is that they could get out of synch with the repertoire and where tags match the natural division of a script into elements like vowels and consonants, having tags serves to document the organization of the script and can make it easier to understand the intent of the rules.

In some situations, declaring some classes explicitly, or using the available set operators defined in [RFC7940] to create classes by combining other ones may make the statement of a particular rule more elegant or easier to understand and verify.

Alternative 3— Evaluating the Whole Label

It is possible to achieve the same restriction without the context rule, that is, by formally writing a rule evaluated over a whole label. In this example, the statement of the rules becomes rather simpler, but it is perhaps more difficult to tell which code point is affected by which context.

```
<!-- same repertoire, but not using "tag" or "when" -->
<data>
 <range first-cp="0780" last-cp="07A5" />
 <range first-cp="07A6" last-cp="07B0" />
 <char cp="07B1" />
</data>

<rules>
  <!-- same set definitions as before -->
  <class name="consonant" comment="set of consonants">
    <range first-cp="0780" last-cp="07A5" />
    <char cp="07B1" />
  </class>

  <class name="vowel" comment="set of vowels">
    <range first-cp="07A6" last-cp="07B0" />
  </class>  <!-- by using "class" instead of "anchor" the two contexts become the
same -->
  <rule name="vowel-follows-consonant">
    <class by-ref="consonant" />
    <class by-ref="vowel" />
  </rule>
  ...
```

To force evaluation of these rules we need a rule that expresses a Thaana label as a sequence of permissible contexts as well as an action that disposes as invalid any label that does not match a sequence of Thaana syllables from start to end:

```
  <rule name="Thaana-Syllables">
    <start />
    <choice count="1+" >
      <rule by-ref="vowel-follows-consonant" />
      <char cp="0782" />
    </choice>
    <end />
  </rule>
  ...
 <action disp="invalid" not-match="Thaana-Syllables" />

</rules>
```

In all the alternatives for theses WLE rules, the restrictions are slightly looser than the full linguistic rule, which allows a “bare” U+0782 only in certain contexts. However, to model the script behavior more faithfully in this regard seems to provide only limited benefit, but with substantial increase in complexity.

Example 4, Syllables in neo-Brahmi scripts

When it comes to the neo-Brahmi scripts of India, the encoding model adopted by the Unicode Standard does not automatically prohibit code point sequences that cannot occur in actual writing. It merely encodes the elements that make up each syllable (akshara), and not the aksharas directly. Labels that contain ill-formed aksharas may not render in a predictable manner. That makes it tempting from a security point of view to strictly limit the label to a sequence of well-formed syllables.

Alternative 1 — Well-formed Syllables

Notionally, a sequence of well-formed syllables can be expressed as a pseudo regular expression as follows:[2]

^Syllable+$

where ^ and $ correspond to the start and end of the label, and Syllable+ indicates one or more occurrences of a valid syllable. We would need to define what constitutes a well-formed syllable. A definition of a syllable, conveniently expressed in Backus-Naur form can be found in [Indic-Layout].

Indic Syllables can be formed using one of three rules

Rule 1: **V[m]**

Rule 2: **{CnH}Cn[v][m]**

Rule 3: **CnH** (at end of label)

Where:

V	independent vowel
v	dependent vowel sign (matra)
m	vowel modifier (Devanagari Anusvara, Visarga, and Candrabindu)
Cn	consonant (with inherent vowel), optionally followed by a Nukta, short for C[n]
H	Halant (or Virama)
{ }	encloses items which may be repeated zero or more times
[ ]	encloses items which may or may not be present

[2] Please note that the syllable constraints shown here are presented as an example to show different methods of representing complex script contexts in WLE rules. The details of the constraints may not correspond to the full complexity of such syllables. Likewise, a real design for an actual LGR supporting one of these scripts may well differ in important ways from the pedagogical examples presented here. (A common reason for deviation from the theoretical model is the fact that different languages may have slightly different constraint, but mixed use may need to be supported).

Let’s look at these rules in turn. Here are examples of all the syllable types formed by Rule 1: **V[m]**

उ = V

0909

उँ = Vm

0909, 0901

Here are examples of some of the syllables that can be formed under Rule 2: **{CnH}Cn[v][m]**

फ = C

092B

फ़ = Cn

092B, 093C

फ़् = CnH

092B, 093C, 094D

फ़ज = CnHC

092B, 093C, 094D, 091C

फ़जी = CnHCv

092B, 093C, 094D, 091C, 0940

Translating this into a single WLE rule seems formidable; therefore we will consider an alternate.

Alternative 2—Reduction to Pair Contexts

From the rules we can read off several simpler restrictions on some of the constituents:

1. H must follow Cn
2. v must follow Cn[3]
3. n must follow C
4. m must not follow H
5. H must precede C or end of label (rule 3)

Note that the first two requirements describe an identical context even though they apply to different sets of code points. If presented as "when" rule both would be implemented as

---

[3] A sequence of type Vv is sometimes recommended, so as to obviate the need to encode yet another independent vowel V’ with a glyph corresponding to that of the sequence Vv. Examples exist in the Bengali script. A complete set of rules would have to go beyond the BNF in [Indic-Layout] in order to account for such exceptions.

```
<rule name="follows-nukta-or-consonant">
  <look-behind>
    <choice>
      <class from-tag="consonant" />
      <class from-tag="nukta" />
    </choice>
  </look-behind>
  <anchor />
</rule>
```

and invoked with a when="follows-nukta-or-consonant" attribute on any code point for a dependent vowel or *halant*. When the rule is evaluated, the <anchor /> would either be a dependent vowel or a *halant* and any label not matching the context condition would be invalid. In the repertoire, the code point for the *nukta* (09C3) would have a tag="nukta", and all consonant code points would need to be tagged with `tag="consonant"` so that they can be referred to in the rule using the <class> element with a "from-tag" attribute as shown above.

An alternative to a single-member class as for the *nukta* is to list the code point directly in the rule

```
<char cp="093C" comment="nukta" />
```

This may be appropriate when the code point is also part of some other larger set, for example, if a rule needs to exceptionally treat one of the consonants. But when the code point forms a class of its own (a nukta is neither a consonant nor vowel nor a halant) then giving it a tag may be more natural.

The next two requirements lead to rules that are slightly simpler:

```
<rule name="follows-consonant">
  <look-behind>
    <class from-tag="consonant" />
  </look-behind>
  <anchor />
</rule>
```

```
<rule name="follows-vowel">
  <look-behind>
    <class from-tag="halant" />
  </look-behind>
  <anchor />
</rule>
```

The second rule assumes that *halant* is tagged with tag="halant". The rule would be invoked with a "not-when" tag, since, according to requirement 4, a preceding *halant* is a prohibited context.

Finally, we need to associate each *halant* with the rule

```
<rule name="precedes-consonant-or-end">
  <anchor />
  <look-ahead>
    <choice>
      <class from-tag="consonant" />
      <end />
    </choice>
  </look-ahead>
</rule>
```

describing the contexts specified by the last requirement (must precede C or end of label).

Compare this set of simple rules to the Indic Structure example in the XML specification [RFC7940] and the reduction in complexity becomes very apparent. Why is this?

The one term of Rule 2 that we have not covered is

{Cn H} C...

which rules out creating syllables from **CnCn** without intervening H, but because **Cn** can occur on its own (without any of the other terms) sequences of consonants not connected by a Halant **Cn Cn** do occur, except not in a single syllable. Since we are not interested in knowing where the syllables begin or end, accounting for the term **{ CnH } C...** is unnecessary.

Conclusion

WLE rules are an invaluable tool for restricting labels that are problematic or potentially risky, but they introduce complexity into the label generation rules for a zone. In designing WLE rules for any zone, it is beneficial to strive for the least complex solution that achieves the required restriction on labels.

For the root zone, in particular, this approach has been mandated in the [LGR-Procedure] by reference to the principles laid out in [RFC6912]. These were originally formulated in reference to the task of repertoire selection, but are nevertheless adaptable and applicable to this purpose.

Techniques to reduce the complexity include

- disallowing a code point by excluding it from the repertoire in order to avoid complex behavior associated with that code point;
- specifying a pair or sequence to limit acceptable sequences to only those enumerated in the repertoire;
- defining simple (usually pair) contexts, commonly for left-hand contexts, over more complex rules modeling full syllables or labels;
- using context rules on sequences to avoid having to build a sequence into a rule;
- and finally, recognizing that not all labels that a user might perceive as ‘malformed’ need to be restricted; some do not pose security issues.

References

[RFC6912] Sullivan, A., Thaler, D. , Klensin, J. , and O. Kolkman, "Principles for Unicode Code Point Inclusion in Labels in the DNS." RFC6912, April 2013, https://trac.tools.ietf.org/html/rfc6912

[RFC7940] Davies, K. and A. Freytag, "Representing Label Generation Rulesets using XML", August 2016, https://trac.tools.ietf.org/html/rfc7940.

[IDNA-XML] Freytag, A., Expressing IDNA2008 Context and Bidi Rules in the XML Format for Representing Label Generation Rulesets, https://github.com/kjd/lgr/blob/master/resources/Expressing%20IDNA%202008%20Contextual%20Rules%20in%20LGR.pdf. Visited 2014-10-28

[Indic-Layout] World Wide Web Consortium (W3C) "Indic Layout Requirements" http://www.w3.org/International/docs/indic-layout/

[LGR-Procedure] Internet Corporation for Assigned Names and Numbers, "Procedure to Develop and Maintain the Label Generation Rules for the Root Zone in Respect of IDNA Labels" (Los Angeles, California, March 2013). https://www.icann.org/en/system/files/files/draft-lgr-procedure-20mar13-en.pdf

[Regex-XML] Freytag, A., WLE Rules in XML Format for Representing Label Generation Rulesets and in Regular Expressions, https://github.com/kjd/lgr/blob/master/resources/regex-to-lgr.pdf. Visited 2014-10-28.