Variant Rules

REVISION 4 - 2014-04-08 - DRAFT

1 Overview

Label Generation Rulesets (LGR) define permissible labels, but may also define the condition under which variant labels may exist and their status (disposition).

Successfully defining variant rules for an LGR is not trivial. A number of considerations and constraints have to be taken into account. This document describes the basic constraints and use cases for variant rules in an LGR by using a notation that is different than the format defined in [XML-LGR]. When it comes time to capture the LGR in a formal definition, the format in this document can be converted to the XML format fairly directly.

From the perspective of a user of the DNS, variants are experienced as variant labels; two (or more) labels that are functionally “the same” under the conventions of the writing system used, even though their code point sequences are different. An LGR specification, on the other hand, defines variant mappings between code points, and only in a secondary step, derives from these mappings the variant labels. For a discussion of this process as it relates to the root zone, see [Procedure].

By using types on the variant mappings and carefully constructing the derivations, the designer of an LGR can control whether some or all of the variant labels created from an original label should be available for allocation (to the original applicant) or whether some or all of these labels should be blocked instead and remain not allocatable (to anyone).

The choice between these alternatives would be based on the expectations of the users of the particular zone, and is not the subject of this document. Instead, this document attempts to point out how to design an LGR to achieve the selected design choice for handling variants.

2 Variant Relationships

A variant relationship is fundamentally a "same as", in other words, it is an equivalence relationship. Now the strictest sense of "same as" would be equality, and for equality we have both symmetry

$$A = B \Rightarrow B = A$$

and transitivity

$$A = B \text{ and } B = C \Rightarrow A = C$$

The variant relationship with its expanded sense of "same as" must really satisfy the same constraint. Once we say A is the "same as" B, we also assert that B is the “same as” A. In this document, the symbol “~” means “has a variant relationship with”. Thus we get

$$A \sim B \Rightarrow B \sim A$$

Likewise, if we make the same claim for B and C (B ~ C) then we do get A ~ C, because if B is "the same" as both A and C then A must be "the same as" C:

$$A \sim B \text{ and } B \sim C \Rightarrow A \sim C$$

3 Variant Mappings

So far, we have treated variant relationships as simple “same as” ignoring that each relationship consists of a pair of reciprocal mappings. In this document, the symbol “→” means “maps to”.

$$A \sim B \Rightarrow A \rightarrow B,\ B \rightarrow A$$

These mappings are not defined between labels, but between code points (or code point sequences). In the transitive case we also get

$$A \sim B \Rightarrow A \rightarrow B,\ B \rightarrow A$$

$$A \sim C \Rightarrow A \rightarrow C,\ C \rightarrow A$$

$$B \sim C \Rightarrow B \rightarrow C,\ C \rightarrow B$$

for a total of six possible mappings. Conventionally, these are listed in tables in order of the source code point, like so

A → B
A → C
B → A
B → C
C → A
C → B

As we can see, each of A, B and C can be mapped two ways.

4 Variant Labels

To create a variant label, each code point in the original label is successively replaced by all variant code points defined by a mapping from the original code point. For a label AAA (the letter “A” three times), the variant labels (given the mappings from transitive example above) would be

AAB
ABA
ABB
BAA
BAB
BBA
BBB
AAC
…
CCC

5 Variant Dispositions

Assume we wanted to allow a variant relation between the letters Ö and O, and perhaps also between Ò and O as well as Ó and O. By transitivity we would have

O ~ Ö ~ Ó ~ Ò

However, we would like to distinguish the case where someone applies for OOO from the case where someone applies for the label ÖÖÖ. In the former case we would like to allocate only the label OOO, but in the latter case, perhaps because people have been used to dropping accents on internet addresses, we would like to also allow the allocation of either the original label ÖÖÖ or the variant label OOO, or both, but not of any of the other possible variant labels, like OOÓ or OÖÓ.

How do we make that distinction?

The answer lies in labeling the mapping O → Ö with the disposition "blocked" and the mapping Ö → O with the disposition "allocatable". In this document, the symbol "x→" means "maps with a blocked disposition" and the symbol "a→" means "maps with an allocatable disposition". Thus:

O x→ Ö
Ö a→ O

When we generate all permutations of labels, we use mappings with different dispositions depending from which code points we start.

In creating an LGR with variants, all variant mappings are always labeled with a disposition. By default, these dispositions correspond directly to the dispositions for variant labels, but as we shall see later, sometimes it is useful to assign intermediate values that have a wider array of sub-types than the final dispositions for the labels.

6 Allocatable Variants

If we start with ÖÖÖ, the permutation OOO will have been the result of applying only mappings with disposition "allocatable" and if we can track which dispositions were used we can allocate both the label OOO and the original label ÖÖÖ.

What this amounts to is deriving a disposition for the variant **label** from the cumulative dispositions of all the variant **mappings** that were used to create the label.

if “all variants” = “a” ⇒ set label disposition to “allocatable”

7 Blocked variants

Blocked variants are not available to another registrant. They therefore protect the applicant of the original label from someone else registering a label that is “the same as” under some user-perceived metric. Blocked variants can be a useful tool even for scripts for which no allocatable labels are ever defined.

If we start with OOO, the permutation ÖÖÖ will have been the result of applying only mappings with disposition "blocked" and if we cannot allocate the label ÖÖÖ, only the original label OOO. This corresponds to the following derivation:

if “any variants” = “x” ⇒ set label disposition to “blocked”

To prevent allocating ÖÓO as a variant label for ÖÖÖ we only need to make sure that the mapping Ö → Ó has been defined with the disposition "blocked" as in

Ö x→ Ó.

8 Pure Variant Labels

Now, if we wanted to prevent allocation of ÖOÖ when we start from ÖÖÖ, we would need a rule disallowing a mix of original code points and variant code points, which is easily accomplished by use of the "only-variants" qualifier, which requires that the label consist entirely of variants and all the variants are from the same set of dispositions.

if “only-variants” = “a” ⇒ set label disposition to “allocatable”

The two code points Ö in ÖOÖ are not arrived at by variant mappings, because the code points are unchanged and no variant mappings are defined for Ö →Ö. In this document, an a “-” means that because of the absence of a mapping there is no disposition. So, in our example, the set of variant mapping dispositions is

ÖÖÖ → ÖOÖ: - a -

but the only-variants conditions requires a a a (no - allowed). By adding a final derivation

else if “any-variants” = “a” ⇒ set label disposition to “blocked”

and executing that derivation only on any remaining labels, we disallow ÖOÖ when starting from ÖÖÖ, but still allow OOO.

Derivation conditions are always applied in order, with later derivations only applying to labels that did not match any earlier conditions, as indicated by the use of “else” in the last example. In other words, they form a cascade.

9 Reflexive Variants

But what if we started from ÖOÖ? We would expect OOO to be allocatable, but the disposition set would be

ÖOÖ → OOO: a - a

because the O is the original code point. Here is where we use a reflexive mapping, by realizing that O is “the same as” O, which is normally redundant, but allows us to specify a disposition on the mapping

O a→ O

with that, the disposition set for ÖOÖ → OOO becomes:

ÖOÖ → OOO: a a a

and the label OOO again passes the derivation condition

if “only-variants” = “a” ⇒ set label disposition to “allocatable”

as desired. This use of reflexive variants is typical whenever derivations with the only-variants qualifier are used.

10 Limiting Allocatable Variants by Subtyping

As we have seen, the number of variant labels can potentially be large, due to combinatorics.

To recap, in the standard case a code point C can have (up to) two types of variant mappings

C x→ X
C a→ A

where a→ means a variant mapping with disposition "allocatable", and x→ means "blocked". By convention, we name the target code point with the corresponding uppercase letter.

Subtyping is a mechanism that allows us to distinguish among different types of allocatable variants. For example, we can define three new types: “s”, “t” and “b”. “s” and “t” are

mutually incompatible, but “b” is compatible with either “s” or “t” (in this case, “b” stands for “both”). With this, a code point C might have (up to) four types of variant mappings

C x→ X
C s→ S
C t→ T
C b→ B

and explicit reflexive mappings of one of these types

C s→ C
C t→ C
C b→ C

As before, all mappings must have one and only one disposition, but each code point may map to any number of other code points.

We define compatibility by our choice of derivation conditions as follows

if “only-variants” = “s” or “b” ⇒ allocatable
else if “only-variants” = “t” or “b” ⇒ allocatable
else if “any-variants” = “s” or “t” or “b” or “x” ⇒ blocked

An original label of four characters

CCCC

may have many variant labels such as the examples listed with their corresponding disposition sets:

CCCC →XSTB : x s t b

This is blocked because to get from C to B required x→. We show the starting label, because variant mappings are defined for specific source code points. The variant label

CCCC →SSBB : s s b b

is allocatable, because the disposition set contains only allocatable mappings of subtype s or b, which we have defined as being compatible. The actual set {s, b} only has two members, but the examples are easier to follow if we list each disposition. The label

CCCC → TTBB : t t b b

is again allocatable, because the disposition set contains only allocatable mappings of the mutually compatible allocatable subtypes *t* or *b*. In contrast,

CCCC → SSTT : s s t t

is not allocatable, because the disposition set contains incompatible subtypes *t* and *s* and thus would be blocked by the final derivation.

The variant labels

CCCC → CSBB : *c* s b b

CCCC → CTBB : *c* t b b

are only allocatable based on the subtype for the C → C mapping, which is denoted here by *c* and (depending on what was chosen for the disposition on the reflexive mapping) could correspond to *s*, *t*, or *b*.

If it is *s,* the first of these two labels is allocatable; if it is *t,* the second of these two labels is allocatable; if it is *b,* both labels are allocatable.

So far, the scheme doesn't seem to have brought any huge reduction in allocatable variant labels, but that is because we tacitly assumed that C could have all three types of allocatable variants *s, t*, and *b* at the same time.

In a real world example, the types *s*, *t* and *b* are assigned so that normally each code point C only has at most one non-reflexive variant mapping labeled with one of these subtypes, and all other mappings would be assigned type *x* (blocked). This holds true for most code points in existing tables (such as those used in current IDN TLDs), although certain code points have exceptionally complex variant relations and may have an extra mapping.

11 Allowing Mixed Originals

If the desire is to allow original labels that are s/t mixed, then the scheme needs to be slightly refined to distinguish between reflexive and non-reflexive variants. In this document, the symbol "r-n" means "a reflexive (identity) mapping of n where the disposition would have otherwise been 'n'". Thus:

if "only-variants" = "s" or "r-s" or "b" or "r-b" ⇒ allocatable
else if "only-variants" = "t" or "r-t" or "b" or "r-b" ⇒ allocatable
else if "any-variants" = "s" or "t" or "b" or "x" ⇒ blocked
else ⇒ allocatable

Doing so allows labels that contain only reflexive mappings (in other words, any original label) to fall through and be allocated in the final derivation.

12 Corresponding XML Notation

The XML format defined in [XML-LGR] corresponds fairly directly to the notation used in this document. A variant relation

C x→ X

is expressed as

<char cp="*nnnn*"><var cp="*mmmm*" disp="blocked" /></char>

where we assume that *nnnn* and *mmmm* are the Unicode code values for C and X, respectively. A reflexive mapping would use the same code point value for <char> and <var> element. Multiple <var> elements may be nested inside a single <char> element, but their "cp" values must be distinct (unless other distinguishing attributes, not discussed here, are present).

A derivation of a variant label disposition

if "only-variants" = "s" or "b" ⇒ allocatable

is expressed as

<action disp="allocatable" only-variants= "s b" />

Instead of using "if" and "else if" the <action> elements implicitly form a cascade, where the first action triggered defines the disposition of the label. The order of action elements is thus significant.

For the full specification of the XML format see [XML-LGR].

13 References

[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: ICANN, March, 2013) http://www.icann.org/en/resources/idn/variant-tlds/draft-lgr-procedure-20mar13-en.pdf

[XML-LGR] Davies, K. and A. Freytag, "Representing Label Generation Rulesets using XML", http://tools.ietf.org/html/draft-davies-idntables-07/.Visited 2014-03-18.