An Alternate Feature Variations Mechanism

Skef Iterum

August 15, 2023

1 Introduction

This is an attempt to flesh out a sketch of a new mechanism for specifying a replacement feature table in a variable font. Behdad Esfahbod and I have been discussing this sketch in OpenFontFormat issue 53

2 Concept

In the current system a single list of condition sets is evaluated in order until one set evaluates all true, and then whatever feature indices are listed in that entry replace the original feature tables. The problem with this system is that one must break down the designspace into small unique areas either geometrically or logically as described in conditions.pdf.

In this new system the basic unit is not feature *tables* but *lookups*:

- Iterate through a list where each element has a condition set.
- Instead of stopping at the first condition set, all entries in the list are evaluated.
- When all conditions in the set evaluate to true, one adds lookups corresponding to some feature from a "true" list (if any).
- When all conditions in the set evaluate to false, one adds lookups corresponding to some feature from a "false" list (if any).

At the end of the search, the total set of added lookups encountered is used as the list for that feature.

3 Tables

The idea here is to add a new lookup variation mechanism mostly analogous to the feature variation mechanism.

3.1 FeatureVariations Table Format 2

Type	Name	Description
uint16	majorVersion	set to 2
uint16	minorVersion	set to 0
Offset32	lookupVariationsOffset	Offset to lookupVariations table, 0 if unused
uint32	featureVariationRecordCount	Number of feature variation records.
FeatureVariationRecord	featureVariationRecords[featureVariationRecordCount]	Array of feature variation records.

3.2 LookupVariations Table

Type	Name	Description
uint32	lookupVariationRecordCount	Number of lookup variation records.
LookupVariationRecord	lookupVariationRecords[lookupVariationRecordCount]	Array of lookup variation records (sorted).

3.3 LookupVariation Record

Type	Name	Description
uint16	featureIndex	The feature table index to match (sort key)
Offset32	lookupAdditionsTable	Offset to a LookupAdditions table

3.4 LookupAdditions Table

Type	Name	Description
uint16	majorVersion	set to 1
uint16	minorVersion	set to 0

Type	Name	Description
uint16	flags	If bit 0 is set start out lookup set with lookups from current feature table
uint32	conjunctionCount	Number of LookupConjunction records.
Offset32	lookupConjunctionRecord[conjunctionCount]	Array of LookupConjunction records.

3.5 LookupConjunction Record

Type	Name	Description
Offset32	conditionSetOffset	Offset to a condition set table
Offset32	trueLookupIndexSetOffset	Offset to a LookupIndexSet table when all conditions are true (0 if unused)
Offset32	falseLookupIndexSetOffset	Offset to a LookupIndexSet table when at least one condition is false (0 if unused)

3.6 LookupIndexSet Table

Type	Name	Description
uint16	lookupIndexCount	Number of LookupList indices in this table.
uint16	lookupListIndices[lookupIndexCount]	Array of indices into the lookup list.

4 Algorithm

The original feature table contains an offset to a `featureParams` table and a list of lookupList indices. The purpose of the algorithm is to allow either or both of these to be substituted in relation to the chosen position in design space.

1. Process the featureVariationRecords in the same way as for a version 1 table
2. For each active feature with a LookupVariationRecord:

a. Allocate an empty feature table structure.
b. Copy any FeatureParams from the current feature table
c. If bit `0` is set, copy list of lookups from current feature table into the set for this feature.
d. For each LookupConjunction record:
    i. If all conditions are true set o = trueLookupIndexSetOffset
    ii. Otherwise set o = falseLookupIndexSetOffset
    iii. Copy each lookup in the LookupIndexSet at o into the set for this feature

5 Requirements

- The initial feature table in `GSUB` for a given tag should be equivalent to the output of the algorithm of that feature for the (format) default instance (all axes `0`).

6 Typical patterns

- A given feature will typically use either the FeatureVariation or the LookupVariation subtable. The exception is if a feature alters its featureParams at points in design space but specifies its lookups with the LookupVariation system.
- When a feature has some lookups used at every point in designspace, but cannot copy those from the initial feature table, they can be added to the set with an initial LookupConjunction Record with a `0` conditionSetOffset. The empty condition set always evaluates to true so the entries in the trueLookupAdditions subtable will always be added.

7 Formal Properties

For the purposes of this discussion assume there is an easy way to logically negate any condition. And for the sake of simplicity ignore the falseLookupIndexSet subtable.

When a feature uses the LookupVariations system the set of lookups will be union of those added for each condition set that evaluates to true. This is analogous to a logical “or”. A condition set evaluates to true when all of its conditions evalute to true, analogous to a logical “and”. Therefore lookups are added to the set based on a disjunction of conjunctions.

Accordingly, the system is "complete" in that any lookup can be included (or excluded) according to any arbitrary boolean formula using the following convention:

1. Convert the formula for each lookup to disjunctive normal form.
2. Pool the conjunctions used among the lookups together.
3. Create a LookupConjunction records corresponding to each individual conjunction with a trueLookupIndexSet containing each lookup that had that conjunction in its DNF.

8 Motivation for false LookupIndexSet table

Although the system is formally complete with the false LookupIndexSet field, which is conceptually analogous to an "else", it simplifies some cases.

GSUB substitutions are typically present or absent, but the variable substitution mechanism can also be used with GPOS. Consider the archetypal GPOS case of a sudden change in kerning between "T" and "o" when the weight (or other axis) makes the latter no longer fit under the former. One can implement that change with clever variable kerning values but one can also implement it by specifying two separate kerning values and switching between them.

When doing the latter, one wants one value used when a specified set of conditions is true and the other in other cases, e.g. when at least one is false. *Positively* expressing the latter without an "else" might take many separate conjunctions (according to how the conditions overlap). The analysis is much easier with the "else".