Lam-Alef and Kashida

Historical approach

Most fonts ligate *lam+alef* (for all relevant *lam*-like and *alef*-like characters) so it is a single glyph. Historically, SIL fonts did *not* do that but used separate lam and alef glyphs, contextually shaped to look like the shape of the traditional lam-alef ligature.

Amiri is another font that takes this approach and this note from one of the Amiri font's fea files hints at potential problems:

```
# Kashida justification as implemented in most, if not all, applications is
# broken by design; it inserts kashidas *after* doing OT layout which makes it
# impossible for OT code to interact with those kashidas breaking all sorts of
# things.
```

One application that breaks is MS Word (at least 2013, 2016): it sometimes inserts a kashida *between* lam and alef when justifying text rendered in our fonts; the resulting rendering is completely unsuitable.

Amiri's designer, Khaled Hosney, goes on to say how he fixed the problem:

```
# To trick LibreOffice/MS Office to not do kashida justification we set the
# default kashida to a zero width, blank glyph which makes them to think there
# is no suitable kashida glyph, resorting back to regular justification
# To get manually inserted kashida working we use an rlig feature to map to the
# actual kashida glyph(s).
```

Because Amiri has curved connections, and thus lots of different kashida glyphs (for different numbers of kashidas inserted), the fea logic is rather complex. For SIL fonts, with flat joins, the solution was much simpler: simply include both an *encoded* zero-width glyph, kashida-ar, and an *unencoded* non-zero width glyph, kashida-ar.haswidth, in our fonts. Then at some point in the fea/gdl, replace kashida-ar with kashida-ar.haswidth.

While this approach did prevent the lam-alef sequences from having visible kashidas inserted in them, it had a significant negative side effect: it prevented Word from using kashida justification where it *was* suitable.

Current strategy

In late 2020 we decided to move away from our existing strategy to one that implemented actual lam-alef ligature glyphs, which of course both fixes the problems and allows applications like Word to use kashida-based justfication. The status of our transition is as follows:

- Scheherazade: implemented in v3.000
- Lateef: implemented in v2.000
- Harmattan: implemented as of v4.000
- Alkalami: has always used ligature glyphs.

Exceptions:

While our strategy is to implement appropriate ligature glyphs for all relevant lam-like and alef-like characters, there is one exception: we have chosen to not create lam/alef ligature glyphs for U+0675 ARABIC LETTER HIGH HAMZA ALEF. This is for two reasons.

- Unicode has discouraged the use of alef with high hamza (U+0675) because the decomposition does not reflect the preferred order of representation.
- High hamza is only word initial and so it is unlikely that a *lam+alef* with *high hamza* would ever occur. Additionally, there is a terrible collision with the vertical position at which we have been asked to

position the high hamza and it doesn't seem worth trying to make it look right when it doesn't occur in the real world.

- Unicode 14.0 encoded 19 new alef characters in the U+0870..U+0883 range. These characters are primarily quranic and it is mostly unknown when they might form a lam-alef ligature. Currently Scheherazade New v3.300 forms a lam-alef ligature with these characters *only* with a *lam* (U+0644) and not with any other lam-like characters. Our other fonts do not implement lam-alef ligatures with the alefs in the U+0870..U+0883 range.

Papers

This is an interesting paper that discusses ligatures (not just lam/alef and lam/lam/heh but other typographic ligatures). It's a very old document, but it discusses some of the problems: Haralambous, Yannis. 1995. The traditional Arabic typecase extended to the Unicode set of glyphs