Devanagari script shaping for type designers

[Draft]

Typography occupies space at the intersection of linguistics, text rendering technologies, and design, and requires some knowledge of all of these areas. This document addresses how Indic (or Brahmic) scripts work, how the Unicode Standard encodes those scripts, what shaping behaviour is expected for rendering, and how OpenType Layout features are utilized to implement the shaping behaviour, especially the Devanagari script.

Scope and intended audience

[The nine Unicode ISCII scripts, with an initial focus on Devanagari.]

Indic characters change shape depending on their context. This document is intended for type designers who designed Devanagari typefaces and wish to turn them into functional fonts. It is also relevant for type users that wish to understand expected shaping behaviour, how the text shaping engines process Devanagari text.

[just an intriguing and welcoming figure]

The intention of this document is to present a tool-independent explanation of the logic and techniques of turning letters and signs into text. Hopefully it will allow to understand the techniques and allow type designers to produce fonts in the tool of their choice. The documentation of expected Devanagari text shaping can be useful for general type users to see if the combination of fonts and text shaping engines they use results in a correct combinations.

This documentation doesn't contain instructions how to draw and construct Devanagari or other letters. Linguistic evaluation of Hindi and other languages, grammar and orthography is also outside of the scope of this project, as we look solely at understanding of text representation of language.

Feedback

The authors will be happy to review feedback and suggestions how to improve this documentation, and we encourage users to open an issue on GitHub. Due to time constrains, however, the authors are unable to help with the production of specific fonts, so if you have such questions, raise them on a public forum such as TypeDrawers.

1 Glossary

[Text representation, encoding, character, text rendering, shaping, glyph, ...]

[Composition vs complex base/sign]

[“Conjoining” instead of “reduced” as it’s less abstract. “Conjoining” as an encoding and shaping term; “dependent” as a term for static analysis.]

[“Written (consonant) cluster” instead of “conjunct”?]

[Letter vs diacritic. Letter character vs combining mark character. Base grapheme (simple letter or complex) vs sign (simple diacritic or complex) grapheme? Base glyph vs mark glyph.]

[Graphemes (base vs sign), glyphs (spacing vs nonspacing); GDEF classes (base vs mark)]

2 Unicode text representation and rendering

Typesetting text was originally about directly reproducing written shapes as typographical glyphs, until digital typography became possible, when exchanging underlying, digitally encoded text became both a possibility and a need.

[Unicode's role in the lifecycle of text: user—keyboard—encoding—display]

In order to separate the concern of meaningful textual units from what exact font is used, the Unicode Standard, as the predominant text encoding technology today, is designed with an architectural separation between abstract *characters* and actual glyphs. So now our exchangeable text consists only of Unicode-encoded characters, while the exact look of these characters is provided by digital fonts that map characters to glyphs.

[comparison between Latin and Devanagari's shaping behavior]

This abstraction is straightforward for scripts like Latin, for example a character U+0041 LATIN CAPITAL LETTER A is largely just mapped to a glyph "A". But for so-called *complex scripts* that are supported by Unicode with their inherent contextual interactions taken into consideration, such as Arabic and Devanagari, the abstraction leads to a contextually dynamic and thus complex mapping between characters and glyphs.

[relationship between text representation and text rendering]

This complex mapping intended by the Unicode Standard is both meant to be the guidelines for *text representation* (i.e., how text is encoded in Unicode character sequences) and expected to be a responsibility of font techonologies for *text rendering*. The process of properly mapping a character sequence to a glyph sequence is known as *text shaping*.

OpenType Layout (OTL) is the de facto standard shaping technology today for implementing Unicode-encoded complex scripts. There are also other shaping technologies, such as AAT (Apple Advanced Typography, available in Apple products) and Graphite (available in LibreOffice, XeTeX, Firefox, etc.), that require differently coded shaping rules to be coded in fonts.

2.1 Unicode Indic encoding principles and shaping requirements

Assamese–Bangla (Bengali), Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Odia (Oriya), Tamil, and Telugu are the nine Indic (also known as Brahmic) scripts that are supported by Unicode with an encoding model based on ISCII-88 (Indian Script Code for Information Interchange, 1988).

[Sanskrit–Devanagari alphabet]

The Unicode ISCII model's behavior exhibits strong influence of both Sanskrit and the contemporary Hindi's Devanagari orthographies. Many other Indic scripts are supported by Unicode with their encoding models more or less derived from the Unicode ISCII model, such as Sinhala (Sinhalese), Myanmar (Burmese), Khmer

(Cambodian), Balinese, and Javanese. While scripts such as Tibetan, Thai, and Lao are encoded with radically different models.

The key characteristics of the Unicode ISCII encoding model are:

- **phonetic segmentation**, thus certain graphically composite characters
- **phonetic encoding order**, thus shaping-stage reordering of certain glyphs
- **vowel killer doubles as conjoiner**, thus shaping-stage discretion for conjuncts
- **conjoiner joins letters to form conjuncts**, thus shaping-stage prioritization of productive conjoining forms

[how characteristics above play roles in text representation and rendering of an akshar]

When analyzing an Indic text's Unicode representation, the text is broken down to a sequence of *akshar* (the classic Indic orthograhic syllable), and Indic-specific complex shaping is only expected within each akshar.

Graphically speaking, each akshar has a single independent base (which can be consonantal or vocalic, simple or complex base) and can optionally have dependent signs (spacing or not) applied (connected or not) to the base.

[Unicode Indic properties]

Phonetic segmentation, phonetic order, and reordering

[Important clarification: phonetic analysis is only an aid to the foundational graphic analysis. Many written structures are not explainable with a classic linear phonetic analysis.]

Each akshar is then linearized and segmented into an ordered sequence of characters according to how Sanskrit systematically identifies the underlying phonetic segments. Phonetically, an akshar can be either /C*VC?/, a vowel optionally prefixed with a consonant (or consonant cluster) and optionally suffixed with one of the special consonants, or /C+/, a pure consonant (or consonant cluster). [And special cases.]

[graphic and phonetic anatomy of an akshar: র্ক্ষাং]

Therefore conjuncts are encoded complexly of basic letters because conjuncts represent consonant clusters. While certain graphically composite structures are encoded atomically because they represent simple phonetic segments.

[The vowel killer sign (virama) suppresses the inherent vowel of a consonant letter, signifying a pure consonant; vowel signs instead alters the inherent vowel to a different one; while vowel modifier signs add additional phonetic value either to or after the nucleus vowel of a living syllable.]

[Decomposition of phonetically encoded atomic split vowel characters.]

[With a generalized analysis, independent vowel letters can be considered as special consonant letters that have a zero consonant and a vowel different from the default inherent one. Dependent vowel signs are these special consonant letters' productive dependent forms.]

Also, the encoding order of an akshar's component characters is thus decided regardless of the writing or visual order. [And special cases.] Therefore certain intermediate glyphs in shaping process are expected to be *reordered* by the shaping engine from the originally encoded phonetic position to the graphic position as they appear.

characters <प ि> → reordered पि

Vowel killer, conjoiner, and conjuncts

As Indic consonant letters are syllabic, including an inherent vowel, pure consonants are marked explicitly with a *vowel killer* sign.

A conjunct is encoded as its phonetically equivalent sequence of independent consonant letters that are to be connected with *conjoiners*. Contextual shaping is thus required for proper rendering. Certain written forms' composition might seem obscure in a specific writing system, but the well-understood Sanskrit orthography often reveals their phonetic structures.

[Typical pure consonant, conjunct, and the interchangeable simplified form]

Traditionally vowel killers are not used inside a phonetic syllable, but in order to write fewer conjuncts, the contemporary Hindi orthography allows a vowel killer to be used inside a syllable to mark a pure consonant when writing certain consonant clusters, and such written forms are largely interchangeable with the consoant clusters' traditional conjunct forms.

This flexibility allows vowel killer characters, such as U+094D DEVANAGARI SIGN VIRAMA (*virama*), to contextually double as the conjoiner in the Unicode ISCII model. Such an encoding model largely leaves the decision of how to form a conjunct to a font's discretion.

characters <क ् ष> → conjoined क्ष

When used for forming conjuncts, the graphic role of being a conjoiner is actually the major reason, and the phonetic role of being a vowel killer often don't make sense.

[Conjuncts that don't make sense phonetically: र्‌क्र · आ्‌ · ৎ্‌]

Because conjuncts are encoded as such virama-connected consonant letters without an explicit hierarchy, conjunct forming rules in fonts need to be prioritized appropriately to enable desired fallback behavior when the whole cluster is not captured by any single conjunct glyph in a font. This is a major source of complexity in shaping.

[example of ZWJ and ZWNJ usage]

A general format control character, U+200D ZERO WIDTH JOINER (ZWJ [zwɪdʒ]), is used in Indic specifically for assisting formation of conjuncts, when a desired productive conjoining form is not shaped by the default behavior. [ZWJ is used on a virama's other side of the manipulated consonant letter. ... between two consonants, immediately next to the virama on either side, so it both prevents the two consonants to form a ... requesting the virama to form a productive conjoining form with the consonant not separated from the virama by the ZWJ, preventing an obscure conjunct.]

Another general format control character, U+200D ZERO WIDTH NON-JOINER (ZWNJ [zwɪndʒ]), is used immediately after a virama to ensure the virama acts only as a vowel killer, without being conjoined with any following character, thus effectively terminating an akshar.

2.2 Text shaping engines

A text shaping engine shapes a Unicode character sequence into a final glyph sequence, with aid of the shaping rules coded in a font. HarfBuzz (open source), DirectWrite/ Uniscribe (Microsoft), Core Text (Apple), and Adobe's unnamed engine are the four groups of OTL shaping engines that matter the most to digital type production.

[table of which engines the commonly encountered products use]

For OTL, the exact shaping operations executed by a shaping engine are decided jointly by the engine's knowledge of the script and the font's rules. In particular, OTL shaping engines try to match a text against any of the predefined, restrictive character cluster patterns, then insert dotted circles as placeholder base glyphs wherever a failure of the match leads to stray marks.

[example of an OTL Indic cluster pattern and a match failure]

Shapers and script tags

How an OTL shaping engine and a specific script's font should work together is prescribed in Microsoft's *script development specifications*, and is implemented in shaping engines conceptually as script-specific *shapers*. An OTL font declares one or more *script tags* to request what shaper should be used.

The OTL shaping behavior for the nine Unicode ISCII scripts have undergone a major change, thus there are the original version (Old Shaping Behavior, sometimes referred to as *Indic1*) and the version 2 (New Shaping Behavior, commonly referred to as *Indic2*), requested with different tags.

script	*original*	*version 2*
Assamese–Bangla	beng	bng2
Devanagari	deva	dev2
Gujarati	gujr	gjr2
Gurmukhi	guru	gur2
Kannada	knda	knd2
Malayalam	mlym	mlm2
Odia	orya	ory2
Tamil	taml	tml2

script	*original*	*version 2*
Telugu	`telu`	`tel2`

Unlike their Indic1 counterparts, Indic2 shapers no longer statically assume a consonant letter to have a particular type of conjoining form. Instead, such a property is defined dynamically by fonts in certain features (rphf, pref, blwf, half, and pstf) in the *basic shaping forms* stage and is retrieved by shapers. Consequently, the reordering process in Indic2 shapers is divided into the *initial reordering* before any shaping rules kick in, and the *final reordering* after the basic shaping forms stage.

For some of the nine Unicode ISCII scripts, their Indic2 shapers are well implemented in major platforms, therefore their Indic1 tags are mostly obsolete and do not need to be supported by fonts. But some still rely on their Indic1 shapers, thus their fonts are recommended to support both Indic1 and Indic2 shapers.

[table of products that still do not support the Indic2 shapers]

[Language tags]

Features and lookups

[...]

3 Required interactions in Indic shaping

Many scripts, such as Latin and Arabic, are conventionally considered to have two basic categories of graphemes, *letters* (spacing and basic) and *diacritics* (nonspacing and modifying) in the context of digital typography. This simple categorization doesn't work well for Indic scripts, as Indic scripts tend to have a rich set of letter-like but conceptually complex structures besides basic letters, and the modifying structures in Indic scripts can be either nonspacing or spacing and are often closely related to basic letters or letter-like complex structures as their dependent forms.

[Latin letters and diacritics vs Indic bases and signs]

Therefore, for analyzing Indic scripts, their graphemes are instead categorized as *bases* (independent structures) and *signs* (dependent and productive structures, known as *mātrā*, *kār*, etc., in various languages).

3.1 Indic bases and signs

Fundamentally, Indic scripts work in a way that, though joining (sometimes not apparent graphically, then only conceptually) a set of basic written units (basic bases and basic signs) together to form compositions that can represent more complicated sounds.

[A two-by-two chart, क कों and ক্ষ ক্ষোং. How ক্ষোং is broken down to a complex base ক্ষ (different from an atomic base क) and several productive signs, and how the whole structure is a composition (different from a simple akshar ক্ষ).]

Compositions are ultimately obscure in terms of their graphical structures, but as an aid to analysis, generalized and productive signs can often be identified in compositions, thus always an akshar analytically consists of a base and optional productive signs. When a productive sign conceptually has a corresponding independent structure, the sign is considered a *dependent form* of the latter.

characters <क ◌ ष> → complex base क्ष

characters <प ◌ र> → complex base? प्र

characters <त ◌ क> → composition त्क

There can be multiple ways to analyze a specific akshar. An analysis may suggest an akshar is either only an obscure base or a composition of a base and signs, and how the signs are identified can differ.

[Alternative analyses for প্র and ট্র: base, base + ra sign vertically (works for both), half sign + alternative base horizontally (only works for প্র)]

Productive signs provide extensibility to a script, and shaping and prioritizing them appropriately provide reasonable fallback patterns for edge case spellings that are out of the intended design scope and are thus not optimized specially. It's beneficial to try to locate an analytical balance between simpler base–sign composition (less contextual

variation, but more glyphs) and more productive signs (fewer glyphs, but more contextual variation).

base ह्र → dependent f͡x

base त → dependent t͡x

base र → dependent x́ or ~x or x̂

[The line between a complex base and a composition (in which there are productive signs identified) is not definite. There can be different analyses. In order to build a font with desired shaping behavior, one appropriate analysis needs to be selected and is implemented in the font as shaping rules.]

Various ways of sign categorization:

- syllabic: virama, vowel sign, consonant conjoining form, syllable modifier, nukta...
- visual relationship: pre-base, below-base, above-base, post-base...
- advance: spacing and nonspacing
- encoding: atomic and complex
- shaping: reordered...
- ...

3.2 Compositions

One more signs can be placed on a base, forming a composition. Due to the encoding principle of phonetic segmentation, some compositions (or part of a composition, such as a split vowel sign) are encoded atomically, such as U+0906 आ DEVANAGARI LETTER AA, which is comparable to Latin precomposed characters such as U+00C1 Á LATIN CAPITAL LETTER A WITH ACUTE. [Normalized shaping process with a decomposing preprocess.]

base <त> → simple akshar त

base and sign <त ◌ं> → composition तं

Both a base and a sign may then have complex encoding, and would require complex shaping internally.

characters <क ◌् ष> → complex base क्ष

characters <र ◌् त> → complex sign र्त

[Technically in a glyph sequence, a composition may consists of multiple GDEF base glyphs and GDEF mark glyphs.]

Reordering

In a glyph sequence, conventionally, pre-base glyphs are stored before a base, while above-, below- and post-base glyphs are all stored after a base. If the encoding order contradicts the expected glyph order, the concerning glyphs generally need to be reordered. Crucial reordering operations are mostly handled by shaping engines.

[Reordering of repha and isign]

[Various reordered signs, atomic or complex.]

Sign carrier

Akshar-level nonspacing (above- or below-base) signs are usually carried by the base, but sometimes another spacing sign or a group of spacing structures jointly are preferred carrier. When the carrier has a significant width, there can often be a preference of where exactly a nonspacing sign should land. The exact positioning on a carrier is control with `abvm` and `blwm`, the OTL GPOS features for *above-base mark positioning* and *below-base mark positioning*, repsectively.

Sign-to-sign interaction

Multiple nonspacing signs can be placed to a carrier at the same side (above or below). Instead of just stacking, they often interact graphically. Such interaction is usually dealt with in `pres`, the OTL GSUB feature originally designed for *pre-base substitutions* but can be used to handle all *presentation forms*.

composition <त ◌े ◌ं ◌̇> → signs interacting तँ

Nukta

Nukta is not an akshar-level sign. Instead, it is placed directly on the grapheme it modifies (typically consonant letters and their dependent forms; certain innovative orthographies apply nukta to vowel letters and vowel signs also). As it is a low-level modifier and a nukta-ed grapheme is generally expected to behave like an atomic grapheme, nukta is generally ligated to the modified grapheme in `nukt`, the OTL GSUB feature for *nukta forms*. [Possibility for GPOS, mark on ligature, and contextual detection.]

[example of creating a nukta form]

3.3 Complex bases (obscure conjuncts)

In a certain analysis, when all the identified productive signs have been stripped away from an akshar, the leftover is a base, either *atomic* (encoded atomically as a single character) or *complex* (encoded complexly as a character sequence). Generally, a complex consonantal base is encoded with conjoiner joining a sequence of basic consonant letters.

Complex bases are formed either in `akhn` or `cjct`, the OTL GSUB features for *akhands* and *conjunct forms*, respectively. As `akhn` presecdes the features for forming complex

signs, while `cjct` is after those, where to form a specific complex base depends on its desired shaping priority and whether the complex base is expected to have its own conjoining forms.

Vowel sign-conjoined complex bases

[রু গু...]

3.4 Complex signs (conjoining forms)

[List how each of the interaction is applicable to each script.]

Complex signs, the productive signs that are encoded as a sequence, are usually conjoining forms of simple or complex consonantal bases, as vowel signs are generally encoded atomically. Conjoining signs are systematically named according to both of their graphic relation and phonetic relation with the base:

- **graphically, conjoined to the base:** *above-base* (initial by default), *pre-base* (leading by default), *below-base* (trailing by default), or *post-base* (trailing by default)
- **phonetically, in the consonant cluster with the base:** *initial* (the first leading consonant), *leading*, or *trailing*

Because the Unicode ISCII model does not resolve conjunct shaping priorities on the encoding level, complex signs especially need to be formed in a prioritized way. Typically, conjoining forms are formed in OTL GSUB features in the following order:

- `rphf`: reph form *(above/post-base initial)*
- `pref`: pre-base form *(pre-base trailing)*
- `blwf`: below-base forms *(below-base trailing)*
- `half`: half forms *(pre-base leading)*
- `pstf`: post-base forms *(post-base trailing)*

When the default prioritization does not shape a desired composition, a ZWJ (U+200D ZERO WIDTH JOINER) is used to override which side of a virama should become a conjoining form.

If a virama is neither used for forming a complex base, nor used by either of its flanking consonants for forming a conjoining form, it does not exhibit the conjoiner behavier and is simply left as a vowel killer on the preceding consonant.

[Dependent forms are conventionally known by various terms: (dependent) vowel/consonant signs (corresponding to independent vowel/consonant letters), superscripts/subscripts, pre/above/below/post-base forms, half forms (either any dependent forms of consonant letters, or specifically the dependent forms of phonetically leading consonants letters that are written as pre-base signs, which are common in scripts such as Devanagari), etc.]

Above/post-base initial (repha)

A special conjoining form of the initial *ra* that is graphically different from other (if any) leading conjoining forms, is conventionally referred to as *repha*. Repha is formed in the OTL GSUB feature `rphf` *(reph form)*, and is subject to reordering in the Unicode ISCII model because it is encoded initially but its attested graphical forms (above- and post-base) all expect a position after the base in a glyph sequence.

characters <र ◌्> त → repha र्त

Script-specific applicability: [Is this ultimately language-specific?]

- **Devanagari and Gujarati:** <ra, virama> BASE → rasignabove (repha) [Or use the linguistic format "ra, virama → rasignabove / _ BASE" instead for a clearer separation of context?]
- **Gurmukhi:** <ra, virama, zerowidthjoiner> BASE → rasignabove (repha), *historical*
- **Assamese–Bangla:** <rabangla/raassamese, virama> BASE → rasignabove (repha)
- **Odia:** <ra, virama> BASE → rasignabove (repha)
- **Telugu:** <ra, virama, zerowidthjoiner> BASE (modern font) or <ra, virama> BASE (historical font) → rasignpostbaseinitial (repha), *historical*
- **Kannada:** <ra, virama> BASE → rasignpostbaseinitial (repha)
- **Malayalam:** <rephdot> BASE → rasignabove (repha), *traditional orthography*
- **Tamil:** *unattested*

Pre-base trailing

Formed in the OTL GSUB feature `pref` *(pre-base form)*, and is subject to reordering.

Script-specific applicability:

- **Devanagari, Gujarati, Gurmukhi, Assamese–Bangla, and Odia:** *unattested*
- **Telugu and Kannada:** BASE <virama, ra> → rasignprebasetrailing, *stylistic*
- **Malayalam:** BASE <virama, ra> → rasignprebasetrailing, *simplified orthography*
- **Tamil:** *unattested*

Below-base trailing

Below-base forms, along with post-base forms, are the predominant consonantal dependent forms in Telugu and Kannada. Formed in the OTL GSUB feature `blwf` *(below-base forms)*, and is subject to reordering in Telugu and Kannada.

Script-specific applicability:

- **Devanagari and Gujarati:** *unattested (rasignbelow, etc., are limited)*
- **Gurmukhi:** BASE <virama, ha/ra/va/...> → ha/ra/va/...signbelow (*additional ones are historical)*
- **Assamese–Bangla and Odia:** BASE <virama, ra/ba> → ra/basignbelow
- **Telugu and Kannada:** BASE <virama, BASE> → khasignbelow/... (khasign)

- **Malayalam:** BASE <virama, BASE> → kasignbelow/..., *traditional orthography*; BASE <virama, la> → lasignbelow, *simplified orthography (limited)*
- **Tamil:** *unattested*

Pre-base leading (half form)

Formed in the OTL GSUB feature `half` *(half forms)*.

[Pseudo half forms that look like dead consonants but are inside the same akshar, telling from the reordered isign and repha.]

Half forms are the predominant consonantal dependent forms in Devanagari and Gujarati. Both simple and complex bases can have half forms. Mainly applicable to typical bases that have a vertical stem on the right side, and the vertical stem is lost in their half forms. [Interchangeable analysis with "a base conjoined by another base below, see: प्त.]

characters <त ◌्> क → half form त्क

Note the consonant letter ra has a true half form (which is also known as an *eyelash*) besides its more commonly used conjoining form for a phonetically leading/initial position, *repha*.

characters <र ◌्> य → repha र्य

characters <र ◌् ZWJ> य → rasignpre ~य

Script-specific applicability:

- **Devanagari:** <BASE, virama> BASE → kasignpre/... (k/...) and <ra, virama, zerowidthjoiner> BASE → rasignpre (r)
- **Gujarati:** <BASE, virama> BASE → kasignpre/... (k/...)
- **Gurmukhi:** <BASE, virama, zerowidthjoiner> BASE → sasignpre, *historical*
- **Assamese–Bangla, Odia, Telugu, Kannada, Malayalam, and Tamil:** *unattested*

[For letters that do not have a half form, virama is shown as a vowel killer. Maintain the consistency between halant forms and half forms for such letters for a smooth experience in typing.]

[Reordering behavior is determined by half forms: ...]

Post-base trailing

Post-base forms, along with below-base forms, are the predominant consonantal dependent forms in Telugu and Kannada. Formed in the OTL GSUB feature `pstf` *(post-base forms)*, and is subject to reordering in Telugu and Kannada.

Script-specific applicability:

- **Devanagari and Gujarati:** *limited*
- **Gurmukhi:** BASE <virama, ya/...> → ya/...signpost (*additional ones are historical)*

- **Assamese–Bangla and Odia:** BASE <virama, ya> → yasignpost (yasign) [Note Odia's ambiguous encoding]
- **Telugu and Kannada:** BASE <virama, BASE> → kasignpost/… (kasign)
- **Malayalam:** BASE <virama, ya/va> → ya/vasignpost (ya/vasign)
- **Tamil:** *unattested*

Devanagari has a semi-productive post-base form of Ya, which appears when there isn't a special complex base form between a stemless consonant and Ya. This form is not shaped as a post-base form in OTL's sense (i.e., in the `pstf` feature), because an OTL post-base form is assumed not to be a vowel sign carrier.

characters <त ◌् > य → half form त्य

characters ट <◌् य> → post-base conjoining form ट्य

3.5 Devanagari-specific knowledge

[…]

4 Language-specific requirements

Every language has its own extension to the baseline Sanskrit usage, while certain graphemes and interactions of the Sanskrit usage can also be obsolete for the langauge.

4.1 Basic character eligibility

Benchmark: Sanskrit

The classic Sanskrit alphabet:

अ आ इ ई उ ऊ ऋ ॠ ऌ ॡ ए ऐ ओ औ

क ख ग घ ङ
च छ ज झ ञ
ट ठ ड ढ ण
त थ द ध न
प फ ब भ म

य र ल व
श ष स ह

with signs ◌ा ◌ि ◌ी ◌ु ◌ू ◌ृ ◌ॄ ◌ॢ ◌ॣ ◌े ◌ै ◌ो ◌ौ ◌ं ◌ः, *a modifier letter* |Avagraha|, *and a marginal sign* ◌ँ.

Various dependent forms and complex bases.

Hindi

Marathi

Nepali

Theoretical completion

4.2 Interaction eligibility

[two-dimensional table between languages and interactions]

- Nukta: Generally used on consonant letters, the sign *nukta* is a secondary modifier that theoretically can actually be applied to any letter and signs. The interaction is often restricted with actual language usage: Hindi: Dda, Ddha; Perso-Arabic: Ka, Kha, Ga, Ja, Pha; English: Ja, Pha; Marathi, Nepali: Ra (Technical consideration: <Ra_Signnukta, Signvirama, <L>> as one of the two ways of encoding R-Deva, if later operations do not consider the <Ra, Signnukta> sequence anymore.); Kashmiri: Ca, Cha, Ja
- Complex bases: CC conjuncts, especially *_ra for stemmed letters; Cv conjuncts such as ra_usign, ra_uusign, ha_rsignvocalic. *Ra: K|Kh|G|Gh|C|J|Jh|Ny|Nn|T|Th|D|Dh|N|P|Ph|B|Bh|M|Y|L|V|Sh|Ss|S|H

- Dependent form: vowel letter -> sign (only trailing; both sides often encoded atomically), consonant letter -> sign (leading, trailing, and coda;) *: K|Kh|G|Gh|C|J|Jh|Ny|Nn|T|Th|Dh|N|P|Ph|B|Bh|M|Y|R|L|V|Sh|Ss|S
- [Sanskrit has a repha form on a vowel base. Not well supported by shaping engines.]

Combined interactions:

- Dependent forms (i.e., half forms) of obscure conjuncts (i.e., CC conjuncts). *R: K|Kh|G|Gh|C|J|Jh|Ny|Nn|T|Th|Dh|N|P|Ph|B|Bh|M|Y|L|V|Sh|Ss|S

5 Typographic considerations

[Side bearings, kerning, size and darkness proportion between Indic and Latin, metrics, Signi (ikar matra) matching, headstroke overlapping…]

A Tutorial on how to build an OTL Devanagari font

This turorial showcases that, with glyphs already in hand, how one can construct OTL rules to build a basic but functional Devanagari font with desired shaping behavior.

[Glyph sequences are structured in way that a base is followed by zero or more combining marks. Preceding characters sometimes are shaped into following marks for easier shaping.]

[Workarounds: Eyelash, reordering,]

[Additional behavior: isign matching, vocalic liquid CV syllable...]

[Character mapping; precomposed characters are needed for certain platforms, although they can be broken down in general shaping.]

[Define the intended scope and glyph set according to the information available in the main text, then list all rules (without omission) in the following sections.]

A.1 Scope

[Devanagari and contemporary Hindi]

[Glyph set]

A.2 Character to glyph mapping

[...]

A.1 OpenType Layout

[...]

```
languagesystem DFLT dflt;
languagesystem dev2 dflt;

feature nukt { ... } nukt;
feature akhn { ... } akhn;
feature rphf { ... } rphf;
feature half { ... } half;

feature pres { ... } pres;

feature abvm { ... } abvm;
feature blwm { ... } blwm;
```

Language systems

```
languagesystem DFLT dflt;
languagesystem dev2 dflt;
```

Nukta forms

Under this feature, we forming *__nukta glyphs so they can participate in later shaping rules like nromal letters.

```
feature nukt {
    lookup nukt.general {

        # Hindi
        sub dda nukta by dda__nukta;
        sub ddha nukta by ddha__nukta;

        # Perso-Arabic, English, etc, loanwords in Hindi
        sub ka nukta by ka__nukta;
        sub kha nukta by kha__nukta;
        sub ga nukta by ga__nukta;
        sub ja nukta by ja__nukta;
        sub pha nukta by pha__nukta;

    } nukt.general;
} nukt;
```

GPOS (blwm) can be used to position nukta on a base instead, then later shaping rules will need to both ignore Signnukta when forming ligatures and position Signnukta on ligature components properly, but also still always recognize the existence of nukta on every component characters (note the ones not next to a virama thus not blocking a virama's function automatically) and decide whether these clusters should shape in a way consistent to their nukta-less counterparts.

Akhand Ligatures

[...]

```
feature akhn {

    lookup akhn.classical_prioritized_complex_bases {
        sub ka virama ssa by k_ssa;
        sub ja virama nya by j_nya;
    } akhn.classical_prioritized_complex_bases;

    lookup akhn.rakar_complex_bases {
        sub ka virama ra by k_ra;
        sub kha virama ra by kh_ra;
        sub ga virama ra by g_ra;
        sub gha virama ra by gh_ra;
        sub ca virama ra by c_ra;
        ...
    } akhn.rakar_complex_bases;
```

```
        lookup akhn.complex_bases {
            sub cha virama ya by ch_ya;
            sub cha virama va by ch_va;
            sub tta virama tta by tt_tta;
            sub tta virama ttha by tt_ttha;
            ...
        } akhn.complex_bases;

    } akhn;
```

Conjoining forms

[OTL pre-base forms are pre-base forms for a phonetically (and thus in encoding) trailing position. ...]

```
    feature rphf {
        lookup rphf.general {
            sub ra virama by repha;
        } rphf.general;
    } rphf.general;

    feature half {
        lookup half.general {
            sub ka virama by k;
            sub kha virama by kh;
            sub ga virama by g;
            ...
        } half.general;
    } half;
```

Presentation forms

[...]

```
    feature pres {
        lookup pres.general {
            ...
        } pres.general;
    } pres;
```

Above- and below-base mark positioning

[probably needs a note what this feature requires, showing anchors, and how they work]

```
    feature abvm {
        lookup abvm.general {
            ...
        } abvm.general;
    } abvm;
```

```
feature blwm {
    lookup blwm.general {
        ...
    } blwm.general;
} blwm;
```

B Tips and commonly used tricks for OTL shaping

[GSUB reordering, IgnoreMarks/MarkAttachmentType, mark tricks (what Andrew G did for Egyptian hieroglyphs)...]

C Tips and reference for glyphs

C.1 Glyphs naming convention

Consistently assigned glyph names facilitate font production.

vocalicrsign

k_ssa.northern-Deva

A simple glyph name is derived from the corresponding Unicode character's name, using the tool, glyphNameFormatter. [something about glyphsNameFormatter, where it can be found, how it works, etc]

An underscore (_) joins multiple glyph names to form a ligature name. Note this is syntax is meant to be used only for true ligatures that have multiple components written in a connected way. Glyphs that are only technically ligatures [what constitutes a technical ligature?], such as glyphs for |ka__nukta|, may use double underscore (__) instead.

A period (.) leads a *variation suffix*, which is always after the main body of a glyph name and before the script suffix.

A dash (-) leads a *script suffix*, which clarifies a glyph's script namespace, and is always placed at the end, after any variation suffixes. The suffix in principle should be the script's code in ISO 15924, *Codes for the representation of names of scripts*. If a script namespace is explicitly set at the project level, that script's suffix can be omitted.

viramasign

rasignabovebase

rasignpostbaseinitial

C.2 Reference glyphs

Devanagari (script suffix -Deva is omitted)

Composition		*Preferred name*	*Alternative names*		*Character sequence*
	अ	**A**		a	•
A Signaa	आ		Aa	a:	• ¬ A Signaa
	इ	**I**		i	•
	ई	**Ii**		i:	• ¬ Ra + I
	उ	**U**		u	•
	ऊ	**Uu**		u:	• ¬ U Signu

Composition		*Preferred name*	*Alternative names*		*Character sequence*
	ऋ	**Vocalicr**		ri	•
	ॠ	**Vocalicrr**		riː	•
	ऌ	**Vocalicl**		li	•
	ॡ	**Vocalicll**		liː	•
	ए	**E**		e	•
E Signe	ऐ		Ai	ai	• ¬ E Signe
A Signo	ओ		O	o	• ¬ A Signo / Aa Signe
A Signau	औ		Au	au	• ¬ A Signau / Aa Signai
	क	**Ka**		ka	•
Ka Signnukta	क़		Qa	(qa)	• ≡> Ka Signnukta
	ख	**Kha**		kʰa	•
	ग	**Ga**		ga	•
	घ	**Gha**		gʱa	•
	ङ	**Nga**		ŋa	•
	च	**Ca**			•
	छ	**Cha**			•
	ज	**Ja**			•
	झ	**Jha**			•
	ञ	**Nya**			•
	ट	**Tta**			•
	ठ	**Ttha**			•
	ड	**Dda**			•
	ढ	**Ddha**			•
	ण	**Nna**			•
	त	**Ta**			•
	थ	**Tha**			•
	द	**Da**			•
	ध	**Dha**			•
	न	**Na**			•
	प	**Pa**			•
	फ	**Pha**			•

Composition	*Preferred name*	*Alternative names*	*Character sequence*
ब	**Ba**		•
भ	**Bha**		•
म	**Ma**		•
य	**Ya**		•
र	**Ra**		•
ल	**La**		•
व	**Va**		•
श	**Sha**		•
ष	**Ssa**		•
स	**Sa**		•
ह	**Ha**	ɦa	•
ळ	**Lla**	(ɭa)	•
क्ष	**KSsa**	kʂa	Ka + Ssa
ज्ञ	**JNya**	ʄɲa	Ja + Nya
क्र ~ ह्र	**[K-H]Ra**	[k-ɦ]ra	[Ka-Ha] + Ra
ह्र	**HaSignvocalicr**	hri	Ha Signvocalicr
◌्	**Signvirama**	<none>	• (contextual)
◌ा	**Signaa**	aː	•
◌ि	**Signi**	i	• (reordered)
◌ी	**Signii**		•
◌ु	**Signu**		• (contextual)
◌ू	**Signuu**		• (contextual)
◌ृ	**Signvocalicr**		• (contextual)
◌ॄ	**Signvocalicrr**		• (contextual)
◌ऌ	**Signvocalicl**		•
◌ॡ	**Signvocalicll**		•
◌े	**Signe**		•
◌ै	**Signai**		•
◌ो	**Signo**		•
◌ौ	**Signau**		•
◌ं	**Signanusvara**		•

Composition	*Preferred name*	*Alternative names*	*Character sequence*
वँ	**Signcandrabindu**		•
वः	**Signvisarga**		•
व़	**Signnukta**		•
र्व	**Signrepha**	Signabovebasera	Ra + <L> (reordered)
क्व ~ स्व	**[K-S]**	Signprebase[K-S]a	[Ka-Sa] + <L>
क्ष्व	**KSs**	SignprebaseKSsa	Ka + Ssa + <L>
ज्ञ्व	**JNy**	SignprebaseJNya	Ja + Nya + <L>
क्र्व ~ स्र्व	**[K-S]R**	Signprebase[K-S]Ra	[Ka-Sa] + Ra + <L>
व्र	**Signrakar**	Signabelowbasera	<L> + Ra

C.3 Stylistic, regional, and orthographic variants

[North variants of letters and digits. Marathi variants of letters.]