UTC L2/21-241
2021-11-20

Khmer Encoding Structure

Executive Summary. This document analyses and describes an extended grapheme cluster structure for Khmer script. It proposes no new characters. The document is being written in close liaison with script experts in Cambodia and undergoing review by staff at various governmental organisations. This document contains a snapshot of the current state of the research to allow for parallel consensus building between experts in Cambodia and the UTC. The purpose is to provide appropriately early engagement with those outside the core team working on this document, while having a proposal mature enough to be worth engaging with.

Issues. A current list of issues for which input is sought is repeated here. From p9 and the structure for Modern Khmer:

- How complex do we want to make the consonant shifter rules? There are some other sequences that we could say: these cause the shifter to downshift. But where do we draw the line? We could make the rules very clever and complicated and thus never need ZWNJ or we can simplify the rules and require more uses of ZWNJ.
- Should ZWNJ come before or after a consonant shifter?

From p22 and Middle Khmer:

- Do we really need ZWJ to push consonant shifters down or is there another analysis for those words?
- Should we differentiate Middle and Modern Khmer syllable structures by language tag?

Of these, the last question is probably of most interest to the UTC and I give more background to this question.

The cluster structures for Modern Khmer and Middle Khmer are quite different. While the structure for Middle Khmer is an extension to that of Modern Khmer, the extensions are pervasive enough that they, if applied to Modern Khmer would cause problems. In particular, Middle Khmer supports vowel sequences and final coengs, that are not supported in Modern Khmer. The costs of implementing support for the Middle Khmer extensions for font developers only interested in Modern Khmer is high. This cost would also apply to all other processing functions. It would also cause problems with people misspelling if not using an appropriate input method. For example:

- while Middle Khmer has some visual distinction between a spacing coeng being stored before or after a non spacing above vowel, this distinction is not made in Modern Khmer (since the final coeng never occurs) and so a user would not necessarily notice the visual distinction and therefore may well store a final coeng for a Modern Khmer word, which they may not notice is wrong.
- The user base for Middle Khmer is very small in comparison to that of Modern Khmer.

The result of these difficulties is that there is a recommendation that Middle and Modern Khmer be given different cluster definitions. The problem is that they they are both the same script and the distinction would be based on language rather than script. This is assumed to be a novel requirement and therefore needs careful review by shaping engine implementers, since they would carry the primary cost of implementation. Notice that the default cluster definition for Khmer script would be for Modern Khmer and only text appropriately language marked would follow the Middle Khmer cluster structure.

Changes. There are various changes that may not be obvious from the document, that are worth highlighting:

- Folding coeng da into coeng ta in Modern Khmer. There is a section on this in the document, but it is a significant encoding change.
- Abandoning the use of ZWJ and ZWNJ to control ligatures in the Muul style. Instead normal font styling is to be used.

Khmer Encoding

Marti)n Hosken (SIL International, NPIC LSDU)

Introduction

The Khmer script has been encoded in Unicode for a long time. But it is a very complex script used to write several languages in that it has multiple orthographies. The primary driving orthography is for the modern Khmer language and there is some confusion in understanding how to encode it. This document is a step in a long history of the discussion of how to encode the Khmer script. It proposes no changes to the character repertoire of the Unicode Standard, but concentrates on the description of what a Khmer sequence consists of and what rules are to be applied to it.
One of the primary concerns with this revision of the encoding is to ensure that **one visual form only has one encoding**. That is, there are not two ways to encode the same visual representation. This is important for three reasons:

1. Confusability. If you can encode the same visual representation in two ways, you can come up with say two different website addresses that look identical, thus allowing someone to spoof another site.
2. Ease of typing. If there are two ways to represent a single visual form, then it is up to the typist to ensure that they enter text in such a way as to get the correct encoding for any given visual form. If, on the other hand, there is only one encoding possible, then the computer can generate that encoding based on a visual input from the typist, regardless of how badly they type that visual form. This means that those who are not experts in Khmer Unicode can successfully enter good data based exclusively on the visual form.
3. Sorting and searching. If there are two ways to encode the same visual representation, people will be confused as to why two identical looking words do not match each other, or sort to different places or why one might be marked as misspelled for no visual reason or why some words cannot be found in a search.

Therefore the current encoding results in significant user experience difficulties (see Appendix 1). Thus while, if one is careful, one can enter Khmer text in a manually enforced consistent fashion with correct rendering, most users are not proficient enough to ensure such consistency of data entry.

When considering the encoded syllable structure for a script there are two extremes that can be created:

- The loose specification aims to allow any possible sequence so long as it meets the above 3 requirements, even if such a sequence could never occur in any orthography using the script. For example, in Khmer, this would allow multiple vowels or multiple non spacing diacritics. The cost with this approach is that implementers then need to support these extra sequences. For example, font developers would need to support vertical stacking diacritics for vowels and other diacritics for Khmer.
- The tight specification aims to only describe sequences that could occur in an orthography. The danger here is that it may limit someone who needs to use the script in unexpected ways for a new orthography or the transliteration of foreign words.

In this discussion, we will start from a relatively tight position of building the syllabic structure from orthographic analysis. Such an analysis cannot be completed without consideration of all orthographies that use Khmer script, including Old and Middle Khmer. In this discussion, ‘Khmer language’ refers to Modern Khmer. Other Khmer language orthographies (like Middle or Old Khmer) are explicitly identified. It is worth noting that modern minority language orthographies have been designed specifically to fit within the structural norms of Modern Khmer. Thus they do not use consonant shifters outside of their use in the Khmer language; only use one vowel character, etc.

In handling what are described as illegal sequences, font developers and shaping engine developers, etc. are free to represent such sequences as they see fit, so long as they are visually contrastive to any other sequence. For example, one developer may decide to insert a dotted circle between two signs, while another may simply stack them.

Rationale

Encoding issues are something that end users of applications, working with Khmer script, should never have to consider. Systems should do the right thing for them, and they should not even need to know what Unicode is. That users need to know about Unicode in order to type is a failing on the part of the system implementers. This is not a criticism because implementation is hard and has lacked the necessary supporting standards. Users should be able to type in a way that is natural to them. They should be able to search for a word and find all the instances of that word across multiple documents, typed by different people with different applications (assuming such a search can deal with the different file formats appropriately).
The current situation, for Khmer, is that users are expected to have a relatively deep understanding of how Khmer is encoded in Unicode in order to type correctly. They are expected to type their coengs, vowels and other diacritics and signs in the right order with nothing to help them visually. It is not a surprise therefore that there can be a plethora of ways in which a single word is spelled (see Appendix 1).
It is assumed, in this document, that the implementation of support for Khmer will include everything needed to hide the encoding complexities from users. There is no need to consider the impact that encoding will have on how end users type. Instead the question is of the impact the encoding has on the implementation of input methods that enable users to type in a natural way without consideration of encoding issues.
One of the purposes of this document, therefore, is to help implementers create solutions that allow users to not have to know about Unicode to be able to type their language correctly. We hope that this document will lead to others that will enable font designers and keyboard implementers to produce fonts and tools that work consistently and enable consistent data entry and rendering. We do this by concentrating on describing an unambiguous encoding structure for Khmer.

One important presupposition of this document is that if there is only one way to encode something, then it makes it easier to produce a system that works with that one way. Thus if there is only one way to store a word, an input method can take a variety of ways that a user might type a word and normalise them into the single correct way. On the other hand, if there are multiple ways to encode a word, then the input method cannot do this and the user is expected to resolve the ambiguity and pick the right ordering.
In the technical context where a keyboard only allows a user to type single codepoints (or short sequences), there has been no other option. Users want to be able to type in a variety of orders and therefore have made use of the existing ambiguities in the encoding. But modern keyboard applications are far more sophisticated and are capable of allowing different typing orders and outputting a single correct order. But they can only do this if there is an agreed single correct order to output. If they are expected to output different orders, then the keyboard cannot resolve its input and the decision of what output to generate has to be pushed back onto users. Users therefore have to become aware of encoding issues. This is the problem this document attempts to resolve: **What is the agreed single correct order to be used?**

Document Conventions

The approach taken in describing the encoding, will be strongly regular expression based both in terms of describing the structure but also in describing transformations for rendering and keyboarding. The regular expression language used is based on PCRE (as used in Perl and Python and a number of other tools). In particular the following more advanced aspects of regular expressions are used:

- Alternation. Sequences are kept together between `|` alternation marks within a group.
- A separately defined subexpression, which is presented as a variable declaration and use, is

considered its own group.

- `(?=`*regex*`)` is a look ahead assertion and `(?<=`*regex*`)` is a look behind assertion. This allows the identification of a specific subexpression in the context of text before and after. This is used in rewrite rules where the matched expression is replaced, leaving the context unchanged.
- `(?!`*regex*`)` is a negative look ahead assertion (fail if the regex matches) and `(?<!`*regex*`)` is a negative look behind assertion. This is not currently used.
- From standard regular expressions we use `?` for optional and `{0,2}` for 'up to two'. Parentheses are used to group sequences and alternations. `[]` match one of the characters in the list.

There are various technical terms that are often used very ambiguously. This document is not immune to such ambiguities. But it is hoped that:

- A **diacritic** is a non spacing character, above or below a letter. This is in contrast to general Khmer linguistics. *Thus for this document the following are also diacritics: non spacing coengs and non spacing vowels.*
- A **symbol** is a character that is not part of a linguistic syllable. For example, digits, punctuation.
- A **sign** is a character that is not a symbol, consonant, coeng, independent vowel or vowel. Most of the signs are diacritics, but some are spacing. Signs may not occur on their own and are part of the syllable structure. This is more commonly called a diacritic in Khmer linguistics.
- The term **spacing**, for example when talking about spacing **signs**, vowels and coengs, is restricted in this document to mean only right spacing, unless otherwise specified. Thus coeng ro (U+17D2 U+179A), coeng vo (U+17D2 U+179C) and prevowels (U+17BE, U+17C1..U+17C3) are not considered spacing. Split vowels with both pre and post components (U+17BF, 1U+17C0, U+17C4, U+17C5) are considered spacing.

The Starting Point

Over the years, there have been a number of attempts at resolving the issues in the Khmer encoding and various orthographic syllable structures have been proposed. The structure proposed in this document follows in that tradition and while it is derived from scratch, the result tries not to stray too far from what has gone before. The starting point in that tradition, is from an unpublished proposal from 2004 [Solá] to refine the Khmer syllable structure:

```
B (R | (ZWNJ? C)) S* (ZWNJ? C)? (ZWNJ? V)? O? (ZWJ S)?
Where
B - Base consonant or independent vowel
R - Robat
ZWNJ - zero width non-joiner
C - Consonant shifter
S - Subscript consonant or independent vowel sign
V - Dependent vowel sign
O - Other signs
ZWJ - zero width joiner
```

This is more complex than is needed for Modern Khmer, given it aims to support Middle Khmer as well. Although, by the time we are finished, the syllable structure will be no less complex. The main issue this description suffers from is that it allows multiple ways of representing the same visual form. In addition, actual font implementations and shaping engines diverge from this description in various ways (e.g. Lindenberg 2019). We will, therefore, build up a new regular expression, but in the same direction. Due to the complexities of Middle Khmer, these will not be considered initially, although they may be mentioned, for this initial development of an encoded orthographic syllable structure. Current minority language orthographies will be considered where appropriate.
All analysis of Khmer encoding so far has concentrated on the rendering of correctly input text with a strong linguistic motivation. Unfortunately, for many Khmer writers, they have insufficient expertise in the linguistics of the language, let alone Unicode, to be sure to enter correctly encoded text. Ambiguities in the encoded syllable mean that it is impossible for tooling to help users to type well. If we can remove the

visual ambiguities from the structure, it is then possible for tooling to enable users to type in any order or resolve common typing errors. It also removes confusability (where two different sequences render identically) which is rife in Khmer encoded data.

Orthographic Syllable Structure

At its simplest, a Khmer orthographic syllable may consist of a single consonant or independent vowel. This is the only required element of an orthographic syllable.

```
Syl = B | I
B = [1780-17A2]
I = [17A5-17A7 17A9-17B3]
```

The characters 17A3, 17A4 and 17A8 are excluded since they are deprecated.
We can treat independent vowels just like base consonants. While they have different linguistics associated with them, visually they behave as bases. For example, it makes no sense for an independent vowel to have another vowel diacritic. But this is purely a spelling issue, there is no visual contrast that needs to be resolved. This also holds for all other diacritics and signs. Thus:

```
Syl = B
B = [1780-17A2 17A5-17A7 17A9-17B3]
```

For the purposes of the syllable structure, there are also other characters that we treat as being a syllable in their own right. They take no diacritics and occur, simply, on their own:

```
O = [17D4-17D7 17D9-17DC 17E0-17F9 19E0-19FF]
```

17D8 is not included because it is deprecated.

A Khmer syllable may also include a single dependent vowel, unlike other scripts of SouthEast Asia, which use vowel sequences:

```
Syl = B V? | O
V = [17B6-17C5]
```

Khmer has subjoined consonants, called “coengs”, in Khmer, that are used for initial consonant clusters (sequence of consonants) and also for syllable chaining (whereby the base consonant is read as the final to the previous linguistic syllable and the coeng as the initial for the next linguistic syllable). In Middle Khmer they can even represent syllable final consonants. These uses are not contrasted and so for orthographic purposes, we treat the final consonant from a previous linguistic syllable that is chained visually with the initial consonant of the next linguistic syllable, as the initial of the next orthographic syllable with the linguistic initial as an orthographic medial. For example, សង្គ្រាម is made up of two linguistic syllables but three orthographic syllables each starting with a base character (ស ង ម). The second base character (ង) in this example is acting as the final consonant of the first linguistic syllable. There may be up to two coengs in an orthographic syllable, (e.g. involving coeng ro or as result of syllable chaining). While, visually, more could occur, two pre-vowel coengs is the limit used in any known orthography. The cost, therefore, of requiring support for more, given they are never used, outweighs the value of increased generality. In Unicode a coeng is made up of a Khmer sign coeng (17D2) followed by a base consonant or independent vowel.

```
Syl = B C{0,2} V? | O
C = 17D2 B
```

Khmer has final consonants, but for orthographic purposes, these are treated as the start of a new orthographic syllable, as mentioned previously. With regard to the ordering of coengs, all coengs have a component that renders under the base or a previous coeng. *Therefore coengs are considered as being ordered downwards from the base character regardless of whether they are spacing.* Fonts need to ensure that contrastive order results in contrastive rendering, e.g. in the contrastive order of coengs in Tampuan:

ក្រ្វា	`U+1780 U+17D2` **`U+179A`** `U+17D2` **`U+179C`** `17B6`
ស្វ្រិ	`U+179F U+17D2` **`U+179C`** `U+17D2` **`U+179A`** `17B7`

Versus the opposite order which would be considered wrong spelling in Tampuan:

ក្វ្រា	`U+1780 U+17D2` **`U+179C`** `U+17D2` **`U+179A`** `17B6`
ស្រ្វិ	`U+179F U+17D2` **`U+179A`** `U+17D2` **`U+179C`** `17B7`

In many fonts coeng ro (179A) has no part that goes directly under the base in such a way that it interacts with another coeng (i.e. the other coeng moves up, down or sideways). In which case the non ro coeng is rendered alongside any part of the coeng ro that goes under the base. But in other fonts, the coeng ro does interact with another coeng. For more discussion on this, see the Shaping and Font Development section. In the Khmer language if there is such a visual interaction, the coeng ro always goes below the other coeng. This is because linguistically, the r comes last in a consonant cluster. But we formalise this and say that coeng ro always comes second in a coeng sequence even if it doesn't sound second.

Consonant Shifter

The consonant shifters, triisap (17CA) and muusikatoan (17C9), are such important signs in Khmer that they require their own very careful analysis. There are contextual shaping issues, keyboarding issues and encoding issues with these signs. This section just considers the encoded syllable issues. For a full discussion of all the issues see the section on Downshifting Consonant Shifters.

In the Khmer orthography it is the orthographic consonant cluster that is interpreted as having a series, even if its constituent consonants are from different series. Where in a sequence should the consonant shifter be stored? There are two schools of thought:

- The consonant shifter immediately follows the consonant it affects. Thus if the shifter changes the series of the base consonant, it is stored after the base and before a possible coeng. Likewise if it affects the coeng, it follows that coeng. The problem is that if the coeng is not spacing, it requires considerable linguistic awareness to decide whether the shifter affects the base or coeng and in some cases not even this is sufficient to resolve the ambiguity.
- The consonant shifter is stored at the end of the consonant cluster, since it applies to the whole cluster.

Since in all orthographies we know about, there is no semantic difference between a shifter before or after a spacing coeng, it is preferable to give the shifter a fixed position in the sequence. Thus we propose the shifter is stored at the end of the consonant cluster.
The later section on consonant shifters examines the whole issue of how in some contexts a consonant shifter changes to the glyph of a -u vowel (17BB). The only structural implications of that section is that it is sometimes necessary to override the default shaping behaviour and to not allow a consonant shifter to 'down-shift'. To do this we introduce the use of zero width non-joiner ZWNJ (200C) to stop a shifter from down-shifting. This affects the syllable description:

```
Syl = B C{0,2} S? V? | O
S = (ZWNJ? [17C9 17CA])
ZWNJ = 200C
```

Simply adding a ZWNJ into the syllable description also introduces an ambiguity. Where a consonant shifter cannot be downshifted (because there is no upper diacritic following), then the ZWNJ is redundant and a string with or without the ZWNJ, by definition, will render the same. To complicate things further, a simplistic algorithm that says any shifter followed by an appropriate above diacritic shifts down, results in ambiguity, since either consonant shifter will result in the same visual representation. Thus, while we can allow any consonant shifter after a consonant cluster, only those that might result in down shifting may take a ZWNJ. The constraints are complicated and we copy them here from the discussion and analysis in the section on Downshifting Consonant Shifters. In copying here we simplify the simple case:

```
S = ( (  (?<=SF CNB{0,2} | B (SCF CNB? | CNB SCF)) ZWNJ 17CA
       | (?<=SS C{0,2} | B (SCS C? | C SCS))       ZWNJ 17C9)
      (?=[17B7-17BA 17DD] | 17B6 17C6)
     |   (?<=SS C{0,2} | B (SCS C? | C SCS))       ZWNJ 17C9 (?=17D0)
     | [17C9 17CA])

BNB = [1780-1793 1795-17A2]        # Consonant no BA [ក-ន ផ-អ]
CNB = (17D2 BNB)
SCF = (17D2 SF)                    # Shifter Coeng First series
SCNF = (17D2 SNF)                  # Shifter Coeng Not First series
SCS = (17D2 SS)                    # Shifter Coeng Second series
SF = [1794 179E 179F 17A0 17A2]    # Shifter First series [ប ឞ ស ហ អ]
SNF = [1780-1793 1795-179D 17A1]   # Shifter Not First series [ក-ន ផ-ឝ ឡ]
SS = [1784 1789 1798 1799 179A 179C 179D]  # Shifter Second series [ង ញ ម យ រ វ ឝ]
```

Signs

In Modern Khmer script, there is only ever one non spacing sign and / or one spacing sign. We categorise the non spacing sign as a modifying sign and the final spacing sign as a modifying final. This gives flexibility to categorise non spacing signs as modifying finals if that is needed (see Middle Khmer).

```
Syl = B R? C{0,2} S? V? MS{0,2} MF? | O
MS = [17C6 17CB 17CD-17D1 17DD]    # Modifying Signs [ំ់៍៎៏័៑៝ ]
MF = [17C7 17C8]                   # Modifying Finals [ ះ ៈ]
R = 17CC                           # Robat [៌ ]
```

Since some diacritics may occur with other diacritics, we generalise the syllable structure to allow two diacritic signs. The specifics are given for each sign.

U+17C6 Nikahit ◌ំ

This marks final nasalisation and, in the Khmer language, occurs in three contexts: on its own, after -u (17BB) and after -aa (17B6). In conjunction with -aa, it pushes a consonant shifter down. May be followed by kakabat or toandakhiat and also samyok sannya in Tampuan.

U+17CB Bantoc ◌់

Bantoc is placed over a final consonant to shorten or change the nature of the vowel in the linguistic syllable (including the inherent vowel) that ends with that consonant. It currently only occurs after a simple base consonant, in Modern Khmer. In Middle Khmer, it may also follow -aa 17B6 where it renders over the consonant (ព្យីភា់ល).

U+17CC Robat ◌៌

This represents a final r on a previous consonant (cf. Repha in Devanagari). It is rarely sounded in Modern Khmer (e.g. ទុជ៌ិន ទុគ៌ិត ទុគ៌ិម) and there is little to be gained in positioning it earlier in the orthographic syllable sequence. It currently only occurs with 17B6 (ា), with 17BB (ុ) or with no vowel. There are legacy transitional issues with this character. All the current syllable structure specifications place robat early in the sequence, before any coengs. This is the only core conflict between valid strings between the various syllable structures. Therefore, we place the robat separately, early in the sequence.

U+17CD Toandakhiat ◌៍

This is a silencer and may occur after any vowel (except 17B6) or final consonant cluster. It is a rare character that, in the Khmer language, most often occurs alone or after 17B7 (ិ) with which it visually ligates.

U+17CE Kakabat ◌៎

This is a + like diacritic that marks exclamation in Khmer. It renders above any vowel. The presence of Kakabat does not cause the shifter to go down, on its own. (See the later section on Downshifting Consonant Shifters). In an extreme case there may be a consonant shifter, vowel and kakabat all stacked up: អ្ឡ៊ី៎ះ. (It can also occur after samyok sannya when transcribing Thai tones.)

U+17CF Ahsda ◌៏

This mark highlights certain consonants as being a discrete word. It may occur after 17C5 (ៅ) in which case it renders centred over the base consonant. Such occurrences are rare but do turn up in the Chuon Nath dictionary.It may also occur before 17C7 (ះ).

U+17D0 Samyok Sannya ◌័

This modifies the vowel and final sound on a syllable. In many respects, samyok sannya behaves like a vowel. It pushes muusikatoan down, but not triisap. Since there is a possible sequence that renders identically with 17BB, samyok sannya may not co-occur with -u 17BB. It has similar positioning behaviour to Kakabat and Ahsda. In Tampuan there are words like: រក្រា័ះ 179A 1780 17D2 179A 17B6 17D0 17C7 where the 17D0 ័() is rendered over the reahmuk.
Samyok Sannya may also be used for transliteration purposes. For example in the Chuon Nath dictionary it may be followed by Kakabat to represent Thai tones.

U+17D1 Viriam ◌៑

This kills the inherent vowel in a consonant and as such marks it as a final. This currently, does not co-occur with Bantoc, which has a similar effect as well. This character is primarily used in transliteration of Old Khmer (Chuon Nath Dictionary), where it is common. It does not co-occur with vowels or shifters.

U+17DD Atthacan ◌៝

This was originally used to indicate the presence of the inherent vowel. In minority languages, its meaning is often reappropriated to represent a vowel, e.g. in Bunong: នហា ៍ /nho ro/ 'unending'. May be followed by kakabat or toandakhiat.

U+17C7 Reahmuk ◌ះ

Acts like a final -h as is found in other scripts in the region (Devanagari, Myanmar). It is spacing but is not considered its own cluster. It can also occur following an independent vowel. There is no expectation to be able to insert a cursor before it. In Tampuan U+17D0 (samyok sannya) renders over the 17C7 even though it is stored before. This is language specific styling. See the example in the discussion of samyok sannya above.

U+17C8 Yuukaleapintu ◌ះ

In the Khmer language, this is a stopped short vowel. In Bunong, on the other hand, this can occur after several vowels ([17B6 17C1 17C2 17C4 17DDា េ ោ៝]), also indicating that the vowel is stopped and short.

Miscellaneous

Deprecated Characters

While deprecated characters should not be used, it is necessary to process them appropriately when they do occur.

`17A3, 17A4, 17A8`	Treat as independent vowels, class `B`
`17B4, 17B5`	Treat as dependent vowels, class `V`
`17D3`	Treat as a modifying sign, class `MS`
`17D8`	Treat as a symbol, class `O`

A number of these characters can be represented using sequences of non-deprecated characters. As such, therefore, those deprecated characters should be considered confusable with the preferred sequences. See the list of formally confusable characters later.

Collation

Collation follows the Chuon Nath dictionary and the Unicode tailoring for Khmer works well pretty much regardless of the order of characters in the cluster. The primary orders are: Consonant, Vowel, Coengs. With Shifter and the Modifiers taking secondary orders. With Coengs being stored before Vowels, the coeng sign character (`17D2`) is given a sort key greater than any vowel, that way Cons + Vowel sorts before Cons + Coeng.

The CLDR tailoring, for Modern Khmer, currently assumes that both robat and coengs follow the consonant immediately. There are a number of direct relations of ro (`179A`) + coeng to the coeng consonant + robat (`17CC`). This also means that shifter cannot go before the coeng otherwise, given that ro (`179A`) can take a shifter, the shifter would interfere with the contraction (a code sequence with a single collation key). This establishes the initial cluster order as:

`B R? C{0,2} S?` or `B C{0,2} R? S?`

Of these two, the first makes the most sense, being closer to existing encoding orders. In addition, it would cause a significant growth in tailoring rules. The cost of changing the order to move the S to before the coeng is significant, if not huge. Notice that this ordering also means that the robat cannot become just another `MS`.

Outstanding Questions

- How complex do we want to make the consonant shifter rules? There are some other sequences that we could say: **these cause the shifter to downshift.** But where do we draw the line? We could make the rules very clever and complicated and thus never need ZWNJ or we can simplify the rules and require more uses of ZWNJ.
- Should ZWNJ come before or after a consonant shifter?
    - ZWNJ after shifter is how people typically type it. Also ZWNJ is seen to naturally break the shifter + vowel sequence and so stop the downshifting behaviour.
    - ZWNJ before shifter makes it easier to replace it with an error character like dotted circle that appears before the shifter rather than the vowel.

Conclusion

After all this analysis, the structure of a syllable may be expressed using a regular expression:

```
Syl = B R? C{0,2} S? V? MS{0,2} MF? | O
S = ( (  (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB SCF))    ZWNJ 17CA
       | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWNJ 17C9)
      (?=[17B7-17BA 17BE 17DD] | 17B6 17C6)
     | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS))   ZWNJ 17C9 (?=17D0)
     | [17C9 17CA])

B = [1780-17B3]                      # Base consonant [ក-ឳ]
BNB = [1780-1793 1795-17A2]          # Consonant no BA [ក-ន ផ-អ]
C = 17D2 B                           # Coeng
CNB = (17D2 BNB)
MS = [17C6 17CB 17CD-17D1 17D3 17DD] # Modifying Sign [ំ-៝   ]
MF = [17C7 17C8]                     # Modifying Final [ះ ៈ]
O = [17D4-17DC 17E0-17F9 19E0-19FF]  # Other: Symbols and digits [។-  ០-   - ]
R = 17CC                             # Robat [៌]
SCF = (17D2 SF)                      # Shifter Coeng First series
SCNF = (17D2 SNF)                    # Shifter Coeng Not First series
SCS = (17D2 SS)                      # Shifter Coeng Second series
SF = [179E-17A0 17A2]                # Shifter First series [ឞ-ហ អ]
SNF = [1780-179D 17A1]               # Shifter Not First series [ក-ឝ ឡ]
SS = [1784 1789 1794 1798-179A 179C 179D]    # Shifter Second series [ង ញ ប ម-រ វ ឝ]
V = [17B4-17C5]                      # Vowel [ា-ៅ]
ZWNJ = 200C                          # Zero width non-joiner
```

The regular expression, for all its complexity, is incomplete in that it allows some illegal sequences:

```
17BB [17D0 17DD]                     # consonant shifter ambiguity
17D2 179A 17D2                       # coeng ro comes last
17CC [17C9 17CA]                     # robat may not occur with consonant shifter
```

While it may be possible to incorporate these into the main regular expression, it would complicate an already complex expression beyond the point of being a useful description. For completeness:

```
Syl = B R? Coengs? S? V? MS{0,2} MF? | O
V = ([17B4-17BA 17BC-17C5] | 17BB (?![17D0 17DD]))
Coengs = ((17D2 [1780-1799 179B-17B3])? 17D2 B)
R = (17CC (?![17C9 17CA]))
```

Downshifting Consonant Shifters

Khmer and any other languages that use consonant shifters, have a contextual behaviour that a consonant shifter, followed by certain above base diacritics, like an above vowel, changes shape to look like a -u vowel (17BB). One might think that the obvious way to address this is to keyboard and store a 17BB instead of a consonant shifter. Whether or not this is a good solution, it is not a preferred option given the technical legacy of data. But storing a shifter is only an option if the two main processes of converting its shape and also of allowing someone to type visually and have a keyboard insert the correct shifter, are both algorithmically definable. This section examines that question: how to render/shape a consonant shifter with an upper vowel and how to convert an input of a -u vowel and upper vowel to the correct consonant shifter.

In Khmer orthography, consonant letters are considered to fall into two series, series 1 and 2. When reading Khmer, the phonetic vowel is derived from a combination of the series of the initial orthographic consonant cluster and the vowel. Most vowel signs have a different sound associated with the two series. Most consonants have series 1 and series 2 pairs. In some cases, where there is no equivalent consonant in the other series a consonant is identified as being able to take a consonant shifter to switch its series. To switch from series 1 to series 2, a triisap (17CA ៊) is used, and from series 2 to series 1, a muusikatoan (17C9 ៉) is used. The set of consonants in series 1 is B1 and the set in series 2 is B2. Within these series, only a small number can take a consonant shifter (B1S, B2S).

```
S = [17C9 17CA]
B1 = [1780 1781 1785 1786 178A 178B 178E 178F 1790 1794 1795 179E 179F 17A0 17A1 17A2]
B2 = [1782 1783 1784 1787 1788 1789 178C 178D 1791 1792 1793 1796 1797 1798 1799 179A 179B 179C
179D]
B1S = [1794 179E 179F 17A0 17A2]
B2S = [1784 1789 1793 1798 1799 179A 179B 179C 179D]
```

Note that 178E (ណ៊) and 179E (ឞ៊) have their series swapped from that implied by the Unicode name in the Unicode chart for Khmer. In addition, 179D (ឝ៊), as used for Pali/Sanskrit transliteration, gave the character series 1. But when the character was reappropriated for minority language use, it was given series 2, in keeping with its visual representation (the 'hair' on the base character does not combine visuallyl with muusikatoan)[1]. Bear in mind that Pali/Sanskrit does not use consonant shifters and so what to do with a downshifted shifter is not an issue. But should such a thing occur, in keeping with the single encoding per representation, where a Pali/Sanskrit transliteration does require a triisap to be downshifted, a muusikatoan should be stored instead. But if the triisap does not downshift, then a triisap is stored. Apart from this exceptional odd spelling, there is no other difficulty. The characters 1793 (ន៊) and 179B (ល៉) do have their other series counterparts, but are included in this list because their other series counterparts are Pali characters, and when showing pronunciation, Pali characters are avoided and a triisap may be used.

But what is the series of a consonant cluster? The series is easy to derive when there is only one consonant involved: the series of the cluster is the series of the consonant. But when a consonant cluster consists of more than one consonant (via coengs) of consonants of different series, which consonant derives the series for the cluster?

Talk about paired series consonants taking consonant shifters and that they can't take a ZWNJ. Their downshifting behaviour is undefined (controlled by the font). Recommendation?

Makara (2016, table 10) presents a chart of orthographic consonant based on a sonority hierarchy such that whichever consonant in a cluster has the lowest sonority in the hierarchy (earlier in the table), that is

[1] The known minority languages that use consonant shifters are: Brao [brb], Jarai [jra], Kuay [kdt], Northern Khmer [kxm], Tampuan [tpu] and they all do downshifting.

the consonant that sets the series for the cluster. His chart is reproduced here using Unicode codepoints. In addition, minority characters are added.

Table 1 - Consonant Sonorities

Class	1st Series	2nd Series
Implosive	ដ(178A) ប(1794)	ឌ(178C) ប៊(1794 17CA)
Unaspirated Plosive	ក(1780) ច(1785) ត(178F) ប៉(1794 17C9) អ(17A2)	គ(1782) ជ(1787) ទ(1791) ព(1796) អ៊(17A2 17CA)
Aspirated Plosive	ខ(1781) ឆ(1786) ឋ(178B) ថ(1790) ផ(1795)	ឃ(1783) ឈ(1788) ឍ(178D) ធ(1792) ភ(1797)
Fricative	ឞ(179E) ស(179F) ហ(17A0)	ឞ៊(179E 17CA) ស៊(179F 17CA) ហ៊(17A0 17CA)
Minor Fricative	ឝ៉(179D 17C9)	ឝ(179D)
Sonorant	ង៉(1784 17C9) ញ៉(1789 17C9) ន៉(1793 17C9) ណ(178E) ម៉(1798 17C9) យ៉(1799 17C9) រ៉(179A 17C9) ល៉(179B 17C9) ឡ(17A1) វ៉(179C 17C9)	ង(1784) ញ(1789) ន(1793) ម(1798) យ(1799) រ(179A) ល(179B) វ(179C)

Downshifting

Typographically rendering a vowel above a consonant with a consonant shifter can lead to visual problems in particular to the line height. For example:

ប៊ឹម

To resolve this, there is an orthographic principle that, in most contexts involving a consonant shifter followed by certain above diacritics, the consonant shifter is rendered as a -u vowel (17BB) instead of its default form. The consonant looks like it has two vowels. The list of diacritics that have this effect are the above vowels and samyok sannya. It may also involve -aa (17B6 ា) followed by a nikahit (17C6 ំ). Before discussing the precise rules for when this contextual rendering occurs, we will discuss how, when reading (or typing) it is necessary to analyse which consonant shifter the -u vowel glyph represents: muusikatoan or triisap.
The consonant shifter is the one appropriate to the series of the consonant cluster that precedes it. For this we use the sonority hierarchy. But we only need to consider those entries involving a consonant shifter. 1794 (ប) has the strange property that 1794 17C9 (ប៉) is not a series shifted consonant. Instead the presence of muusikatoan changes the nature of the consonant (to a PA). 1794 17CA (ប៊) is series shifted. But, counterintuitively, it is only the 17C9 (៉) that is 'down shifted' following BA 1794 (ប), and 17CA (៊) never shifts down following BA (1794 ប). This can be addressed by saying that we ignore 1794 17CA (ប៊) and treat PA 1794 17C9 (ប៉) as a simple consonant shifted consonant. In removing the BA, PA and all single codepoints from the above Table 1, we see that a number of cells become empty (see Table 2). In addition, the plosives and fricatives have only 2nd series entries while the minor fricative and sonorants have only first series. Thus we can condense the chart into two sets and one negative set that lists all bases not in S1B. These lists will be important in calculating which shifter a -u form represents, which we will come to.

Table 2 - Consonant Sonorities simplified

Class	1st Series	2nd Series
Implosive		

Unaspirated Plosive		អ៊(17A2 17CA)
Aspirated Plosive		
Fricative		ឞ៊(179E 17CA) ស៊(179F 17CA) ហ៊(17A0 17CA)
Minor Fricative	ឝ៉(179D 17C9)	
Sonorant	ង៉(1784 17C9) ញ៉(1789 17C9) ន៉(1793 17C9) ម៉(1798 17C9) យ៉(1799 17C9) រ៉(179A 17C9) ល៉(179B 17C9) វ៉(179C 17C9)	

```
SF = [179E 179F 17A0 17A2]            # Shifter First series
SNF = [1780-179D 17A1]                # Consonants not in SF
SS = [1784 1789 1793 1794 1798-179D]  # Shifter Second series (including BA/PA)
SCF = (17D2 SF)                       # Shifter Coeng First series
SCNF = (17D2 SNF)                     # Coengs not in SCF
SCS = (17D2 SS)                       # Shifter Coeng Second series
```

Notice that BA 1794 (ប) has been added back into the second series set since 17C9 is pushed down following BA. This is contrary to the linguistics around BA, and is a pragmatic engineering solution to the issue of handling BA. 1793 (ន) and 179B (ល) are included for historic reasons (Chuon Nath, 1967).

Shaping

In shaping text, the need is to know how to render a sequence of Unicode characters. In the case of downshifting consonant shifters, we can write transform rules in shaping the text:

```
(?<=B R? C{0,2}) [17C9 17CA] (?=VA) → 17BB
```
where
```
VA = [17B7-17BA 17BE 17D0 17DD] | (17B6 17C6)
```

17BF, 17C0 (ឿ,ៀ) are not included in the VA class above, because their sound is not dependent on the series of the consonant. In conjunction with PA the shifter does not go down. 17D0 (័) and 17DD (៝)[2] are added because they are diacritics that act like vowels in this context, although samyok sannya (17D0) only pushes a muusikatoan down. The sequence 17B6 17C6 (ាំ) involves a nikahit that renders over the vowel but still pushes a shifter down. Notice that the 17BB in the rule above is not an actual -u vowel stored in data, it is an internal representation (glyph) of a 17BB, for example in a corresponding glyph string inside a font. We also need to bring back the special case of BA which means that not all consonant shifters shift down. The rules become more refined:

```
(?<=1794 C{0,2}) 17CA (?=VA) → 17CA              # no action rule, not strictly necessary
(?<=1794 C{0,2}) 17C9 (?=VA) → 17BB
(?<=BNB R? CNB{0,2}) [17C9 17CA] (?=VA) → 17BB
where
BNB = [1780-1793 1795-17A2 17A5-17A7 17A9-17B3]   # Base No BA
CNB = (17D2 BNB)
```

Since the transform applies to all consonants, a rendering implementation needs to show a visual contrast between a downshifted consonant shifter and the illegal sequence of two vowels: 17BB VA (or VA 17BB).

But there is a big problem with this algorithm in that it makes no difference whether a consonant is followed by a muusikatoan or a triisap, the shifter will downshift. This means that two different sequences result in the same visual representation. We have to disambiguate the expression and only downshift

[2] Used as a vowel in Bunon [brb] (where it marks the following consonant as having the inherent vowel) and Central Mnong [cmo].

those sequences that would result if we converted from the downshifted form to the unshifted sequence. Only one of the two shifters should downshift in any context. To derive these rules, we need to analyse which consonant shifter a down shifted form represents. This is akin to the question of which shifter should be inserted if someone types visually a 17BB followed by a VA (or equivalently from OCR).

Text Entry

On the opposite side of the question is that of how to convert from a visual representation to the underlying Unicode text. What happens if someone types both a 17BB and an above vowel? This is an illegal sequence in the structure. But it is indicative that the user is typing according to the visual representation of a consonant shifter and an input method can help to achieve the result desired by the user. The only real issue is that we don't know which consonant shifter to replace the 17BB with, to transform what is currently an illegal sequence into a legal one. Unlike the shaping rules, which while strongly motivated by the Khmer orthography, apply structurally to all strings, the text entry issue is much more tied to the Khmer orthography. The rules presented here probably work for other languages but may just as well need to be changed.
Again, transform rules can help here. We assume that the 17BB is in the string preceding the VA, for the purposes of the transformation, even though no data should be stored like this.

```
(?<=SF R? C{0,2} | B R? SCF C? | B R? C SCF) 17BB (?=VA) → 17CA
(?<=SS R? SCNF{0,2} | SNF R? SCS SCNF? | SNF R? SCNF SCS) 17BB (?=VA) → 17C9
```

The first rule says that if there is a first series consonant anywhere in the consonant cluster then the 17BB becomes a triisap. Likewise the second rule says that if there is no first series consonant in the consonant cluster but there is a second series consonant, then the 17BB becomes a muusikatoan. (N.B. First and second series consonants here means those that take a consonant shifter to change series, not those that have a corresponding consonant in the other series. Also consonants include coengs here.)
What happens if a user types a different consonant than is in the SF or SS classes? The answer is implementation dependent, since it is up to the system to, if possible, only generate structurally correct text. Thus an error may be indicated by beeping or, if the text is passed through, the rendering system will show a suitable contrast. In the case of a Middle Khmer keyboard, it is highly unlikely that it would be visually based in this way, but if it were, it could insert ZWJ.

Structure

Now we are in a position to specify unambiguously, which sequences downshift. This also has the side effect of specifying for which sequences a ZWNJ (which indicates that the shifter should not downshift) may occur. The structure is similar to the rules for mapping from -u to a shifter. We need to identify whether a shifter will be downshifted. This can occur if a triisap occurs after any first series consonants or a muusikatoan occurs following any second series, if there are no first series consonants in the cluster. To aid with reading, the key new elements in a regular expression are highlighted.

We allow 'bad sequences' such as a first series consonant followed by a muusikatoan, even though this is against the spelling rules of Khmer. Since they are visually distinct, we do not need the font to mark a distinction by making the sequence illegal.

```
S = ( (  (?<=SF R? C{0,2} | B R? SCF C? | B R? C SCF)             ZWNJ 17CA       ❶
       | (?<=SS R? SCNF{0,2} | SNF R? SCS SCNF? | SNF R? SCNF SCS) ZWNJ 17C9)      ❷
      (?=[17B7-17BA 17BE 17D0 17DD] | 17B6 17C6)                                  ❸
     | [17C9 17CA])                                                               ❹

SF = [179E 179F 17A0 17A2]                  # Shifter First series
SNF = [1780-179D 17A1]              # Shifter Not First series
SS = [1784 1789 1793 1794 1798-179D]  # Shifter Second series (including BA/PA)
SCF = (17D2 SF)                     # Shifter Coeng First series
SCNF = (17D2 SNF)                           # Shifter Coeng Not First series
SCS = (17D2 SS)                     # Shifter Coeng Second series
```

The regular expression is addressing the question of what sequence is legal. If there is no ZWNJ in the string, then the regular expression is simple, it is just line ❹ and `[17C9 17CA]`. If there is a ZWNJ in the sequence then the regular expression required to match it for legality is more complicated. Now we read the regular expression:

1. The sequence ZWNJ 17CA may only follow a consonant cluster containing a first series consonant that takes a shifter. Without the ZWNJ the triisapp would be downshifted. So the addition of the ZWNJ keeps the shifter from downshifting and is visually contrastive.
2. The sequence ZWNJ 17C9 may only follow a consonant cluster containing a second series consonant that takes a shifter, so long as there is no first series consonant in the cluster. Without the ZWNJ the muusikatoan would be downshifted. So the addition of the ZWNJ keeps the shifter from downshifting and is visually contrastive.
3. These two sequences may also only occur if followed by an upper vowel.
4. Otherwise there can be any shifter.

BA

But there is a constraint and that is that if there is a BA in the cluster, then any triisaap (17CA) is not going to downshift even if there are other first series consonants that might take a 17CA that would downshift. This has to happen because if the 17CA downshifted then it would be indistinguishable from the BA 17C9. To achieve this, we need to remove coeng BA from the first leg, which matches the consonant cluster with a first series in it. To do this we remove the BA from the C (Coeng) class.

```
BNB = [1780-1793 1795-17A2]           # Consonant no BA
CNB = (17D2 BNB)
S = ( (  (?<=SF R? CNB{0,2} | BNB R? (SCF CNB? | CNB SCF))    ZWNJ 17CA
       | (?<=SS R? SCNF{0,2} | SNS R? (SCS SCNF? | SCNF SCS)) ZWNJ 17C9)
      (?=[17B7-17BA 17BE 17D0 17DD] | 17B6 17C6)
     | [17C9 17CA])
```

In addition the regular expression is slightly restructured to reduce its length. There is no functional difference.

Samyok Sannya

Samyok sannya (`17D0`) only pushes down musikatoan (`17C9`). It does not push a triisaap (17CA) down. This is a common contrast and is probably worth expressing in the regular expression. But there may be other such rules. The question is whether the added complexity is worth the cost of requiring an extra ZWNJ in some sequences.

```
S = ( (  (?<=SF R? CNB{0,2} | BNB R? (SCF CNB? | CNB SCF))    ZWNJ 17CA
       | (?<=SS R? SCNF{0,2} | SNF R? (SCS SCNF? | SCNF SCS)) ZWNJ 17C9)
      (?=[17B7-17BA 17BE 17DD] | 17B6 17C6)
     | (?<=SS R? SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS))      ZWNJ 17C9 (?=17D0)
     | [17C9 17CA])
```

Robat

Robat never occurs with a consonant shifter, so we can remove it from the regular expression. The result is the final expression that is sufficient for Modern Khmer.

```
S = ( (  (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB SCF))    ZWNJ 17CA
       | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWNJ 17C9)
      (?=[17B7-17BA 17BE 17DD] | 17B6 17C6)
     | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS))   ZWNJ 17C9 (?=17D0)
     | [17C9 17CA])

BNB = [1780-1793 1795-17A2]                   # Consonant no BA
CNB = (17D2 BNB)                              # Coeng no BA
```

```
SCF = (17D2 SF)                              # Shifter Coeng First series
SCS = (17D2 SS)                              # Shifter Coeng Second series
SF = [179E-17A0 17A2]                        # Shifter First series
SS = [1784 1789 1793 1794 1798-179D]         # Shifter Second series (including BA/PA)
```

Middle Khmer

There is a later section that analyses Middle Khmer and the structure of the orthographic syllable that is necessary. Here we just discuss possible extensions to the consonant shifter structure that Middle Khmer may need.

ZWJ

Middle Khmer introduces the need for adhoc downshifting where a ZWJ is used to force a downshift where one would not normally occur. But this means that while a ZWJ can precede any consonant shifter, it cannot follow a downshifting shifter. This complicates the structural regular expression.

```
S = (  (  (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB? SCF))   ZWNJ 17CA
                       # contains a first series and no BA => 17CA goes down
        | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWNJ 17C9
                       # contains a second series => 17C9 goes down
        | (?<=SNF SCNF{0,2}) ZWJ 17CA               # not first series stays up ❶
        | (?<=SNS SCNS{0,2}) ZWJ 17C9)              # not second series stays up ❷
       (?=[17B7-17BA 17BE 17DD] | 17B6 17C6)        # if followed by upper vowel
     | (  (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS))  ZWNJ 17C9
        | (?<=SNS SCNS{0,2}) ZWJ 17C9)              # not second series stays up ❸
       (?=17D0)                                     # if followed by samyok sannya
     | (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB? SCF)) ZWJ 17CA
       (?=[^17B6-17BA 17BE 17DD] | 17B6 [^17C6])    # first series elsewhere stays up ❹
     | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWJ 17C9
       (?=[^17B6-17BA 17BE 17D0 17DD] | 17B6 [^17C6])       # second series stays up ❺
     | [17C9 17CA])                                 # or no ZW(N)J

SCNS = (17D2 SNS)                            # Shifter Coeng Not Second series
SNS = [1780-1783 1785-1788 178A-1793 1795-1797 179B 179E-17A2]      # Shifter Not Second series
ZWJ = 200D
```

Within the context of an upper vowel that downshifts shifters, if the consonant context is not first series, then the triisap will stay up and a ZWJ can be used to push it down ❶. Likewise for second series consonants ❷. If the downshifting is caused by a samyok sannya then second series would be downshifted and so non second series would stay up and a ZWJ will downshift them ❸. What about clusters with first and second series but outside the context of a downshifting diacritic (upper vowel or samyok sannya)? In ❹ we match the consonant cluster context for a downshifting triisap, but the ‘vowel’ context is the negative of what would cause downshifting. Likewise for second series, where we also include the absence of samyok sannya ❺.

Inversion

The complexity of this regular expression is such that it may be easier to implement negatively. What would the rules look like if we were to accept the basic sequence `((ZWJ|ZWNJ)? [17C9 17CA])` and then use replacement rules to replace any illegal `ZWJ` and `ZWNJ` with `25CC` (dotted circle)?

```
(?<=SNF R? SCNF{0,2}) ZWNJ (?=17CA [17B7-17BA 17BE 17D0 17DD] | 17B6 17C6) → 25CC
(?<=SNS R? SCNS{0,2}) ZWNJ (?=17C9 [17B7-17BA 17BE 17DD] | 17B6 17C6) → 25CC
                      ZWNJ (?=[17CA 17C9] ([^17B6-17BA 17BE 17D0 17DD] | 17B6 [^17C6]) → 25CC
                      ZWNJ (?=17C9 17D0) → 25CC
(?<=SF R? CNB{0,2} | BNB R? (SCF CNB? | CNB SCF)) ZWJ (?=17CA [17B7-17BA 17BE 17DD] | 17B6 17C6) →
25CC
(?<=SS R? SCNF{0,2} | SNF R? (SCS SCNF? | SCNF SCS)) ZWJ (?=17C9 [17B7-17BA 17BE 17D0 17DD] | 17B6
17C6) → 25CC
```

Middle Khmer

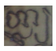
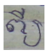
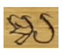
In order to arrive at a syllable structure that can support all data needs, it is necessary to analyse all the orthographies that use Khmer script. For the most part, the minority language use of the Khmer script is conservative with few innovations. Any such innovations are discussed in the main text. But there is one other major set of orthographies that use the Khmer script and these are often lumped under the simplified names of Old Khmer and Middle Khmer, even though they cover many different orthography and language stages in the development of Khmer from Old Khmer in the 7th-8th century, Angkor Khmer and Middle Khmer. Upon examination of these different orthographies, it is the Middle Khmer period, leading into the Modern Khmer period, that requires the greatest complexity in encoding. During this period, there is a growth in orthographic complexity along with little or no standardised spelling.
The concern here is to ensure that they can be adequately encoded. Again the principles of a single encoding for a single graphical representation are necessary. This carries with it an implication that the encoding does not need to directly represent the linguistic structure of the syllable. For example, syllable final coengs (following a vowel) do not necessarily have to be encoded after the vowel.
The first step in such an analysis is to find examples of words that do not fit the currently proposed Khmer encoded syllable structure and see what they say to us. There may be a way to make them fit into the current model or they may call for a change of the model. Care must be taken that the needs of historic data do not overwhelm the far more prevalent modern data. The conversation needs to be one of mutual respect and concern.

Final Coengs

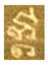
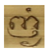
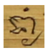
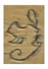
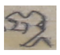
Final coengs occur after the vowel as a final sound in a linguistic syllable and they are often visually indistinguishable from prevowel coengs. There are some contexts in which the final coeng is visually contrastive with a before vowel coeng and some not. The table shows - if data exists and is not visually contrastive between the coeng occurring before or after the vowel, and + if it is visually contrastive. ? Indicates no examples have been found.

Coeng/Vowel	Left	Left Above	Left Right	Right	Above	Below
Coeng Left	-	?	?	-	-	-
Coeng Below	-	-	?	+	-	?
Coeng Right	-	-	+	+	+	+

We examine some examples in detail:

1. Where the vowel is spacing, final spacing coengs are clearly contrastive. Consider:  ហ្លា្យ `17A0 17D2 179B 17B6 17D2 1799`. Thus we do need to encode spacing final coengs after most vowels. Spacing coengs have no visual contrast following pre vowels.
2. In Modern Khmer, a spacing coeng may take a vowel over it, indicating the vowel comes after the coeng: ព្យី `1796 17D2 1799 17B8`. In Middle Khmer, the vowel may render over the consonant, indicating the spacing coeng comes after the vowel:  ពី្យ `1796 17B8 17D2 1799`. Therefore there is a visual contrast between an initial and a final spacing coeng with an above vowel. Where the coeng is initial, the vowel occurs over the coeng, slightly to the right of the base. For a final coeng the vowel is positioned directly over the consonant. Not all fonts show this, but that is because in modern Khmer there is no contrast necessary. But for a Middle Khmer font, such contrast is required.
3. For lower vowels (-u, -uu) the visual contrast is akin to that between multiple coengs, where the -u/-uu vowel interacts just like a non-spacing coeng would in a coeng sequence:  មូ្យ `1798 17BC 17D2 1799`. For comparison, the corresponding Modern Khmer encoding of this sequence would

be: ម្យូ `1798 17D2 1799 17BC`, which looks different and would sound different. Therefore, we cannot simply say that the Modern Khmer spelling with the coeng before the vowel is sufficient.

4. There are final coengs that are indistinguishable from the corresponding coeng before the vowel:  កី្ដ `1780 17B8 17D2 178A`, which is visually indistinguishable from the corresponding Modern Khmer encoding: `1780 17D2 178A 17B8`. There is one base for which there is a visual distinction:  ញំ្ង `1789 17C6 17D2 1784` contrasts with the Modern Khmer encoding: ញ្ងំ `1789 17D2 1784 17C6`. Typically the lower swash under the base is removed in Modern Khmer when a coeng occurs after it. But this swash is not removed if the coeng comes after a vowel.
5. With a spacing vowel, a non-spacing coeng may or may not have contrastive positioning. For example:   ទាំ្ង `1791 17B6 17C6 17D2 1784`. These two examples are not contrastive with the second being a stylistic variant of the first and so the Modern Khmer encoding of `1791 17D2 1784 17B6 17C6` could be used. But  បា្ទ `1794 17B6 17D2 1791` is contrastive and so a contrast is necessary.

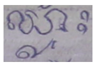
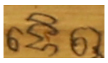
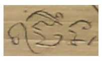
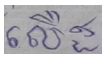
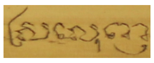
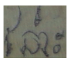
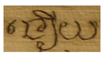
It is evident that number 1 above requires a contrastive encoding. Also number 4 has no visual contrast and therefore requires there not be different encodings. Whether contrastive encodings should be used for numbers 2 and 3 is dependent on regular usage. Most Khmer fonts show a visual contrast between a coeng before or after a non spacing vowel. Number 5 has the confusion of whether a contrast is necessary. It is reasonable to expect this context to show a visual contrast. One complexity is that there can be no visual contrast for a spacing coeng before or after a pre vowel [`17C1-17C3`].

Throughout this analysis, our aim is to have only one encoding for one visual representation. This complicates the syllable description, but is essential:

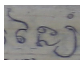
```
CR = 17D2 [1783 1788 178D 1794 1799 179E 179F 17A1]           # Right spacing coeng
Syl = B R? C{0,2} S? (VP MS? | VF MS? C? | V? MS? CR? DS?) | O
VP = [17C1-17C3]
VF = [17B6 17C4 17C5]
V = [17B7-17C1]
```

Multiple Vowels

While Unicode has encoded all the modern Khmer vowels as units, there are vowel combinations in Middle Khmer that are not available as a unit in Unicode.
These need reviewing since most of them only have one occurrence and come from OCR[3].

3.	ឈ្លុាះ	1788 17D2 179B **17BB 17B6** 17C7	
4.	ទេិព	1791 **17C1 17B7** 1796	
5.	ច្រេឹន	1785 17D2 179A **17C1 17B9** 1793	
6.	លេឺង	179B **17C1 17BA** 1784	
7.	ស្រលេុញ	179F 17D2 179A 179B **17C1 17BB** 1789	
8.	នែាះ	1793 **17C2 17B6** 17C7	
9.	នៀីយ	1793 **17C0 17B8** 1799	

[3] See Valy D., et al.

12.	នៃ្យំ	1793 **17C3** 17D2 1799 17C6	
13.	ពា្យាប់	1796 **17B6** 17D2 1799 **17B6** 1794 17CB	

In considering the first 3 examples (1-3) involving a spacing vowel, the question is whether any positional contrast is required for the diacritic vowel in the sequence. The first two examples look nearly identical and do not call for any visual contrast. Likewise the third example looks to not be careful in its diacritic vowel placement, implying its position not carrying meaning. Thus a fixed order is sufficient and storing the non spacing vowel before the spacing vowel is most natural and fits with existing shaping engines. The other examples are simply pre vowel plus diacritic sequences for which there is no corresponding split vowel code. Again a simple fixed order of pre vowel first will suffice.

We analyse examples 3 and 7 as downshifted consonant shifters. But the sequences do not contain the necessary above diacritic to require downshifting. Therefore, there needs to be a way to indicate that downshifting should occur when it would not happen automatically. If ZWNJ can be used to inhibit downshifting, then ZWJ can be used to cause it. But again, ZWJ may only be used in the opposite context to ZWNJ, where downshifting would not occur. The impact on the structure is examined in the section on Downshifting Consonant Shifters.

The last examples (12-13) are noticeable because, in effect, they have two syllables in one, with two vowels separated by a coeng. This raises the spectre of fully chained syllables. Whereby a new full syllable is built on a final coeng. There are various options in how to address this:

- Fully recursive definition whereby the final coeng is also treated as the initial of another full syllable. Thus FinalC would be something like `CF Rhyme?`.
- Chain syllables by allowing the final coeng sign character (`17D2`) to hang as the last of the previous syllable, then its consonant becomes the start of the next syllable.
- List out all the possible options for a limited single extra syllable.

The first option is hard to implement. Recursive regular expressions are nigh on impossible to work with. The second option is merely a way of removing the recursion while keeping the full chaining. The third option is easiest to implement but only supports a very limited following syllable structure with no further chaining. Given there is no evidence of any more extensive chaining than the last example above, we choose the simplest option. For this we allow final spacing coengs to take some limited vowel or signs.

Chuon Nath uses a double pre vowel `17C1 17C1` to represent the corresponding Thai vowel. There are two options on how to handle this:

1. Add 17C1 17C1 to the syllable structure
2. Encode a new codepoint for the double letter as per Thai.

The first is a minimal engineering cost. The second involves a full proposal. While both are possible and acceptable solutions, solution 1 is probably the simpler, given that it doesn't involve changing existing data or fonts.

Combining vowel sequence constraints with final coeng constraints and the regular expression starts to become rather complex..

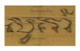
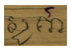
```
Syl = B R? C{0,2} S? Rhyme MF? | O
Rhyme = VP? (VA | VB)? VF MS? FinalC?        # spacing vowel => any final coeng
      | Vsplit (VA | VB)? MS? FinalC?        # spacing vowel component => any final coeng
      | (VP? (VA | VB) | VPA) MS? CR?        # diacritic vowel => spacing final coeng
      | (VP | VO | VPP)? MS?                 # no final coeng
      | (?<=1789 S?) VA CN                   # special case (not drop flourish)
FinalC = (CR (VA | VB | VF)? MS? | CN S?)

CR = 17D2 [1783 1788 178D 1794 1799 179F]    # Right spacing coeng [ឃ ឈ ឍ ប យ ស]
CN = 17D2 [1780-1782 1784-1787 1789-178C 178E-1793 1795-1798 179A-179E 17A0-17A2]
```

```
                                            # Non spacing coengs [ក-គ ង-ជ ញ-ឌ ណ-ន ផ-ម វ-ឞ ហ-អ]
VP = [17C1-17C3]                                    # Pre vowel [េ ែ ៃ]
VA = [17B7-17BA]                                    # Above diacritic vowel [ិ ី ឹ ឺ ]
VB = [17BB-17BD]                                    # Below diacritic vowel [ុ ូ ួ]
VF = [17B6]                                         # Following spacing vowel [ា]
VO = [17BF 17C0]                                    # Split vowel with coeng yo final component [ឿ ៀ]
Vsplit = [17C4 17C5]                                # Split vowel with final vowel component [ោ ៅ]
VPA = 17BE                                          # Pre + diacritic vowel [ើ ]
VPP = 17C1 17C1                                     # Doubled vowel from Thai
```

The VO category consists of split vowels that have a final component that originated from coeng yo. They do not take any other final coeng.
Notice that we do not allow both an upper and lower vowel together, since that is used to visually mark a consonant shifter in conjunction with an upper vowel.
It may be tempting when faced with all this complexity to simply say: allow VP? (VA | VB)? VF? | VO and simplify the rhyme. But the aim is to ensure only one way to encode one visual sequence and to do that we need to disallow sequences that look identical to other sequences.

Diacritics

Robat

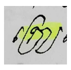
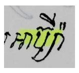
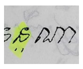
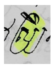
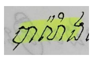
There is some disagreement about where Robat (17CC) renders in relation to the -aa vowel (17B6):   សូត៌ា 179F 17BD 1782 17B6 17CC, which does not help in deciding whether the robat should go early in the sequence or late.

Triisap

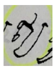
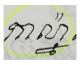
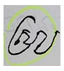
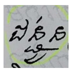
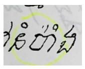
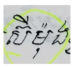
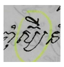
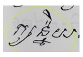
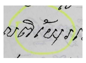
In addition to its function as a consonant shifter, in Middle Khmer triisap may also represent the consonant ង (1784 NGO) super joined above another base. This is akin to robat or kinzi in Myanmar script. Thus សឃ៊ (179F 1783 17CA) may be written more conventionally as សង្ឃ (179F 1784 17D2 1783).

More Examples

Handwritten 1899 Luke[4]

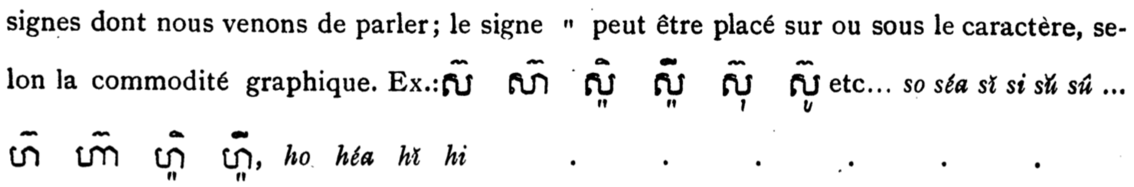
lk2a		ក្រោ្យ	1780 17D2 179A 17C4 17D2 1799	
lk2b		អាប៊ី្យា	17A2 17B6 1794 17CA 17B8 17D2 1799 17C9 17B6	Syllable chaining with muusikatoan on the ‘final ya’.
lk3a		ន៉ណា	1793 200D 17C9 178E 17B6	Variant downshifted muusikatoan.
lk3b		ស្រី្យ	179F 17D2 179A 17B8 17D2 1799	
lk4		បាឈ៉ាំង	1794 17B6 179B 17C9 17B6 17C6 1784	Muusikatoan on ឈ and stays up

4 BFBS 1899

lk6		ឥ្យ	17A5 17D2 1799	
lk8		កាល្ន៉ា	1780 17B6 179B 17D2 1793 17C9 17B6	
lk43a		ស្រា្យ	179F 17D2 179A 17B6 17D2 1799	
lk43b		ជន្ជូ៌ន	1787 1793 17D2 1787 17BC **17CB** 1793	Novel context for bantoc with a double acting final.
lk50		ប‌៉ាំង	1794 **200C** 17C9 17B6 17C6 1784	Muusikatoan stays up
lk63		ស‌៊ីម៉ុង	179F **200C** 17CA 17B8 1798 17C9 17BB 1784	Triisap stays up
lk104		ស‌៊ី្យ	179F **200C** 17CA 17B8 17D2 1799	Triisap stays up
lk127		កូន្ន៉ើយ	1780 17BC 1793 17D2 1793 17C9 17BE 1799	Muusikatoan stays up
lk311		ពិយ៉ែរ	1796 17B7 1799 17C9 17C2 179A	Muusikatoan goes down

Of particular note is lk43b. The bantoc (U+17CB) is rendered above the second base consonant to indicate that it is the final of the syllable and that, therefore, the following coeng is the start of the next syllable. Linguistically, therefore, the Bantoc comes before the coeng. But visually it comes after the sequence of base + coeng + vowel, in keeping with its position as a diacritic in the syllable structure. While this is not the most convenient place for it, linguistically, it is in an unambiguous position and its function, therefore, is clear. It marks the base consonant as a final.

lk3a shows an alternative form of the pushed down muusikatoan. The basis for this analysis comes from Bernard (Bernard, 1902) in which he states:



signes dont nous venons de parler; le signe " peut être placé sur ou sous le caractère, selon la commodité graphique. Ex.: ស៊ ស៊ា ស៊ិ ស៊ី ស៊ុ ស៊ូ etc... *so séa sĭ si sŭ sû* ...

ហ៊ ហ៊ា ហ៊ិ ហ៊ី, *ho héa hĭ hi* . . . . . .

An approximate translation states that the muusikatoan is sometimes placed above or below the character according to the graphical context. This implies that the lower double vertical stroke is merely a stylistic variant of a downshifted consonant shifter (since either shifter may resolve to this form) and is therefore in the realm of font variation. The problem here is that there is no upper vowel in this example. So either this is another vowel (or doubled -u), or it is a phantom downshifted shifter for which we need a way of encoding, perhaps using ZWJ. Until further analysis resolves the quandary, we assume ZWJ.

Khom Thai

The first writing system used for writing Thai was based on Khmer script. The 'Khom' (Thai for Khmer) Thai script has a long history which is not of concern here. The issue is how Khom Thai may be encoded using the Khmer Unicode block. It seems most natural to encode Khom Thai using Khmer characters due to the similarities in the writing systems. Many of the characters look very similar and Khom Thai has coengs which the Thai script does not. The main extensions needed are:

- Addition of Thai tones as diacritics, to Khmer. This can either be done through encoding extra characters or by allowing the sharing of Thai tone marks to be shared with the Khmer script using script extensions.
- Extra vowel combinations and/or extra characters.

There are a number of characters needed to encode Khom Thai in Unicode using the Khmer block. Analysis is required for these. As it stands a further Unicode proposal will be necessary to enable Khom Thai to be adequately encoded in Unicode. Such a proposal will also need to include any changes to the Middle Khmer orthographic syllable structure to ensure it facilitates the encoding of Khom Thai.

Outstanding Questions

- Do we really need ZWJ to push consonant shifters down or is there another analysis for those words?
- Should we differentiate Middle and Modern Khmer syllable structures by language tag?

Conclusion

The final all encompassing syllable structure is:

```
Syl = B R? C{0,2} S? Rhyme MF? | O
Rhyme = (  VP? (VA | VB)? VF MS? FinalC?      # spacing vowel => any final coeng
         | Vsplit (VA | VB)? MS? FinalC?      # spacing vowel component => any final coeng
         | (VP? (VA | VB) | VPA) MS? CR?      # diacritic vowel => spacing final coeng
         | (VP | VPP| VO)? MS?)               # no final coeng
FinalC = (CR S? (VA | VB | VF)? MS? | CN S?)
S = (  (  (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB? SCF))   ZWNJ 17CA[5]
                      # contains a first series and no BA => 17CA goes down
        | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWNJ 17C9
                      # contains a second series => 17C9 goes down
        | (?<=SNF SCNF{0,2}) ZWJ 17CA                 # not first series stays up
        | (?<=SNS SCNS{0,2}) ZWJ 17C9)                # not second series stays up
       (?=[17B7-17BA 17BE 17DD] | 17B6 17C6)          # all if followed by upper vowel
      | (  (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS)) ZWNJ 17C9  # second series goes down
        | (?<=SNS SCNS{0,2}) ZWJ 17C9)                # not second series stays up
       (?=17D0)                                       # both if followed by samyok sannya
      | (?<=SF CNB{0,2} | BNB (SCF CNB? | CNB? SCF))      ZWJ 17CA
       (?=[^17B6-17BA 17BE 17DD] | 17B6 [^17C6])      # first series elsewhere stays up
      | (?<=SS SCNF{0,2} | SNF (SCS SCNF? | SCNF SCS))    ZWJ 17C9
       (?=[^17B6-17BA 17BE 17D0 17DD] | 17B6 [^17C6])       # second series stays up
      | [17C9 17CA])                                  # or no ZW(N)J

B = [1780-17B3]                               # Base consonant [ក-ឳ]
BNB = [1780-1793 1795-17A2]                   # Consonant no BA [ក-ន ផ-អ]
C = 17D2 B                                    # Coeng
CR = 17D2 [1783 1788 178D 1794 1799 179F]     # Right spacing coeng [ឃ ឈ ឍ ប យ ស]
CN = 17D2 [1780-1782 1784-1787 1789-178C 178E-1793 1795-1798 179B-179E 17A0-17A2]
                                      # Non spacing Coeng (excl rho)
                                      # [កខគ ងចឆជ ញដឋឌ ណតថទធន ផពភម  លវឝឞ ហឡអ]
CNB = (17D2 BNB)                              # Coeng no BA
MF = [17C7 17C8]                              # Modifier Final [ ះ ៈ ]
MS = [17C6 17CB 17CD-17D1 17D3 17DD]          # Modifier Sign [ំ់៍      ]
O = [17D4-17DC 17E0-17F9 19E0-19FF]           # Symbols and digits [។-  ០-  ]
R = 17CC                                      # Robat
S = [17C9 17CA]
SCF = (17D2 SF)                               # Shifter Coeng First series
SCNF = (17D2 SNF)                             # Shifter Coeng Not First series
```

[5] Character sequences that are not contextual are highlighted to support easier reading of the regular expression.

```
SCNS = (17D2 SNS)                             # Shifter Coeng Not Second series
SCS = (17D2 [1784 1789 1798-179A 179C 179D])    # Shifter Coeng Second series [ង ញ មយរ វ ឝ]
SF = [179E 179F 17A0 17A2]                    # Shifter First series [ឞ ស ហ អ]
SNF = [1780-179D 17A1]                        # Shifter Not First series [ក-ឝ ឡ]
SNS = [1780-1783 1785-1788 178A-1793 1795-1797 179B 179E-17A2]      # Shifter Not Second series
                                      # [កខគឃ ចឆជឈ ដឋឌឍណតថទធន ផពភ ល ឞសហឡអ]
SS = [1784 1789 1794 1798-179A 179C 179D]     # Shifter Second series [ង ញ ប មយរ វ ឝ]
VA = [17B7-17BA]                              # Above Vowel [ិ ី ឹ ឺ ]
VB = [17BB-17BD]                              # Below Vowel [ុ ូ ួ ]
VF = [17B6]                                   # Following Vowel [ា ]
VO = [17BF-17C0]                              # Split vowel with coeng final [ឿៀ ]
VP = [17C1-17C3]                              # Pre Vowel [ េ ែ ៃ ]
VPA = 17BE                                    # Pre + diacritic vowel [ើ]
VPP = 17C1 17C1                               # Doubled vowel from Thai
VS = [17B7-17BA]                              # Downshifting vowels [ិ ី ឹ ឺ ]
Vsplit = [17C4 17C5]                          # Split vowel with final vowel [ោៅ ]
ZWNJ = 200C                                   # Zero Width Non-Joiner
ZWJ = 200D                                    # Zero Width Joiner
```

This expression is way more complex than the one in the introduction. But it has one major benefit. This expression will not match two strings that should look the same. There is only one match for one visual form. Strictly speaking, though, this is not entirely correct and the few exceptional cases are covered in the next section on confusables where an added list of banned sequences is given.

Exceptions

The syllable definition is not complete in that some visually identical strings get through. The following regular expressions describe sub sequences within a syllable, that the syllable definition should not pass:

```
17BB (17D0 | 17B6 17C6)              # consonant shifter ambiguity (Middle Khmer)
17D2 179A 17D2                       # coeng ro comes last
17C1 [17B6 17BA]                     # units as sequences (Middle Khmer)
17CC [17C9 17CA]                     # robat may not occur with consonant shifter
```

Confusability

There are two levels of confusability with regard to the Khmer script. There is formal confusability where two different strings render identically. Then there is visual confusability where due to the small size of visual features, users often confuse one character or cluster for another. This is particularly prevalent with non spacing coengs.

Coeng Da (‘្ដ’ U+17D2 U+178A) and coeng Ta (‘្ត’ U+17D2 U+178F)

In modern Khmer script, coeng da and coeng ta look identical. One might expect that users know which one to enter and there should be no problem. But, as Appendix 1 shows, confusion is rife. Users regularly type the wrong one. Since they look identical, it is not possible for a user to see how they have mistyped just by looking at the text. This issue arises every so often in the Khmer press: "Method for distinguishing coeng da and coeng ta to avoid confusion" and “Beware confusing coeng da and coeng ta".
One solution would be for a keyboard input method to analyse the context and store the right coeng even if a user typed the wrong one. The problem is that the rules are too complex, with too many special cases for a keyboard to do that.

Because there are two ways of encoding the same visual representation here, we need to resolve this. The easiest solution is to just use a single underlying code for the two coengs. There are no minimal contrasts between the two coengs and therefore there is no need for contrastive encoding. Since both coengs look identical, it is impossible for someone looking at a text to tell whether two different encodings or a single encoding has been used. When it comes to keyboard entry, the keyboard still allows users to type a coeng da or coeng ta as before. The only difference is that it stores one of the two possible encodings, and the user is none the wiser. The user does not have to change their behaviour and they have the advantage that they can’t type the wrong one by accident.

The preferred encoding is that both coeng da and coeng ta are stored as `17D2 178F` (coeng ta). The storage of coeng da (`17D2 178A`) is not disallowed in Middle Khmer where there is a visual contrast. But where the text is known to be Modern Khmer, coeng da should never be stored, in favour of coeng ta. Keyboards for Modern Khmer should only output coeng ta and never coeng da. For Old or Middle Khmer where there may be a visual contrast, then both may be used, but fonts need to show a visual contrast between the two.
Other processes working on Khmer text, such as spelling checkers and text-to-speech systems, will have to be aware of the linguistic ambiguity that has been introduced by folding the two characters together.

Formally and Informally Confusable

Khmer script has many opportunities for visual similarity, especially between coengs. These similarities often confuse users and it can help if a keyboard can resolve them where appropriate. We class these as informally confusable sequences. In addition, there are formally confusable characters and sequences which should not occur and are best marked as erroneous. In effect they are part of the syllable description as being invalid, but it would make the regular expression impossibly complex to exclude them there. So we keep a separate list below. Formally confusable characters and sequences imply a canonical equivalence between the two confusables. But canonical equivalence implies that both sequences may be stored. The point here is that the confusable sequence should never be stored and so be marked as erroneous, specifically to distinguish it from something that it might be presumed to be canonical equivalent to.
Informally confusable sequences, on the other hand, may occur, but should not, and that some visual distinction is needed between the sequence and what it is confusable with.
The list lists sequences that must or should not occur in Khmer text. For example `1791 17D2 1794` should not occur, instead `17A1` is recommended. In this case it is not structurally illegal but a font would do well to show some visual distinction, with the former sequence perhaps not looking as good as the single character. On the other hand `17BB 17D0` may not occur and a shaping engine should show an error when it

occurs.

```
# Formally confusable
17C1 17B8            # use 17BE (កេី=កើ)
17C1 17B6            # use 17C4 (កេា=កោ)
17D2 179A 17D2       # coeng rho must be the last coeng
17BB 17D0            # samyok sannya causes downshifting so cannot occur with -u (មុ័=ម័ុ)
17D2 178A            # Map to 17D2 178F (not in Middle Khmer)

# Informally confusable
17A3                 # deprecated and confusable with 17A2
17A4                 # deprecated and confusable with 17A2 17B6
17A8                 # deprecated. Use 17A7 1780
17B4                 # deprecated invisible character. Remove
17B5                 # deprecated invisible character. Remove
17E2 17D3            # deprecated. Use 19E0
17D8                 # deprecated. Use 17D4 179B 17D4 (។ល។)
1791 17D2 1794       # confusable with 17A1 (ទ្ប=ឡ)
17BE 17D2 1799[6]      # confusable with 17BF (កើ្យ=កឿ)
17C1 17B8 17D2 1799  # confusable with 17BF (កេី្យ=កឿ)
17D2 1799 17BE       # confusable with 17BF (ក្យើ=កឿ)
17C1 17D2 1799       # confusable with 17C0 (កេ្យ=កៀ)
1794 17D2 1789       # confusable with 17AB (ប្ញ=ឫ)
17AD 17B6            # confusable with 1789 (ឭា=ញ)
17AE 17B6            # confusable with 1789 (ឮា=ញ)
1796 17D2 1789       # confusable with 17AD (ព្ញ=ឭ)
1796 17B6 17D2 1789  # confusable with 1789 (ពា្ញ=ញ)
1796 17D2 178B       # confusable with 17B0 (ព្ឋ=ឰ)
[178A 1791] 17D2 178B # confusable with 17D2 1792 (ដ្ឋ=ដ្ធ ទ្ឋ=ទ្ធ)
1796 17D0? 1793 17D2 178B    # confusable with 17D2 1792 (ពន្ឋ=ពន្ធ ព័ន្ឋ=ព័ន្ធ)
[17AA 17B3] 17D2 1799 # confusable with 17B1 17D2 1799 (ឪ្យ=ឱ្យ ឳ្យ=ឱ្យ)
17A7 [17B7 17CC 17CD] # confusable with 17B1 (ឧិ=ឧ៌=ឧ៍=ឱ)
178A 17D2 1792       # confusable with 178A 17D2 178B (ដ្ធ=ដ្ឋ)
1789 17D2 179C       # confusable with 1796 17D2 179C 17B6 (ញ្វ=ព្វា)
```

Yuukaleapintu (‘ៈ’ U+17C8) vs colon (‘:’)

These two characters are very similar and widely misused. We believe that confusability for spoofing is less of an issue as punctuation cannot typically be used in places where spoofing is a problem (e.g. urls, file paths).

To help distinguish, in common use of colon in Khmer, users insert a space before the colon. The keyboard should do this automatically to help assist users in avoiding ambiguity.

[6] The use of final coengs here, presupposes the later analysis from Middle Khmer.

Transition

Having a shiny new syllable structure is all very well, but there is a lot of existing data already in existence and systems based around the various existing Unicode orthographic syllable structures specified over various versions of the standard. Will this be just another such structure, to be ignored or half implemented by systems implementers and what is to be done with existing data?

Compatibility

There are two approaches that can be taken in transitioning correct data to the new structure:

- Require data to be transformed because currently correct data is not correct in the new structure.
- Change the new structure, if needed, to ensure that existing correct data does not need to be transformed.

The latter is much preferred, because requiring the re-encoding of all existing data would be an unacceptably costly activity. Therefore we need to compare the new structure with the existing. We only need to do this for Modern Khmer and minority orthographies and so we can take a simplified view of the new structure:

```
SylN = B R? C{0,2} S? V? MS? MF? | O
```

There are a number of existing syllable structures both in standards and implementations. We examine them in terms of the classes and structures already defined. The mappings are not precise, but sufficient for our analysis. SylE is the specification from the introduction. SylU is the current Unicode standard (v14). SylMS is the Microsoft Khmer shaper specification. SylH is the Harfbuzz Khmer shaper implementation.

```
SylE =  B (R | (ZWNJ? SC)) C* (ZWNJ? SC)? (ZWNJ? V)? (MS | MF)? (ZWJ C)?
SylU =  B (R | SC) (C R?)* (Z? V)? (MS | MF)? C?
SylMS = B C{0,2} (VP | VB)? SC? VA? MS? VF? MF?
SylH =  B Z? (SC | R)? (C (Z? (SC | R))?)*
        (ZMS* VP? ZMS* VB? ZMS* (Z? VA)? ZMS* VF? ZMS* C? MF* | 17D2)
Z = [200C 200D]                    # ZW(N)J
ZMS = (Z* MSNR)                    # ZW(N)J plus modifier sign
R = 17CC                           # Robat
SC = [17C9 17CA]                   # Shifter Character
```

From the description above, the SylMS syllable structure should not support the strings that its implementation actually does. We assume that the specification is out of date with respect to the implementation. Lindenber (2019) provides a more detailed survey of how implementations actually validate Khmer syllables, at that time.

In order to decide whether we can transition without having to transform existing good data, we need to examine the intersection between these syllable structures and the one defined earlier. Are there required strings that are not in that intersection? There are a number of places:

- Unicode doesn't currently allow a consonant shifter after a coeng. But it is unique in that.
- The Microsoft shaper allows a consonant shifter only after a pre or below vowel. This is a very strange place to put one.
- ZWNJ is stored before the consonant shifter.

Incompatibilities

Shifter order: very often a shifter is stored before the coeng when it is now required after.

Canonical Combining Classes

Khmer gets off lightly with regard to non zero canonical combining classes (CCC). These are problematic because once set they can never be changed and so any sequences that would be reordered during normalization due to canonical combining classes have to be reanalyzed and the syllable structure changed to ensure that normalized strings, stored in combining canonical order, conform to the syllable structure.
The only two characters with non zero CCC values are U+17D2 (Coeng) CCC=9 and U+17DD (Atthacan) CCC=230. Since U+17D2 and U+17DD are never adjacent as specified in the syllable structure, there is no problem.

Comparing with Current Unicode

Discuss things like zwnj before vowels.

Normalization

Here we describe normalization actions (not normalization as defined in UAX #15) where sequences in currently conformant Unicode orders are folded into a single sequence using the structure described here.

Reference Implementation

The enclosed reference implementation normalizes Modern Khmer text to the encoding described here in such a way that it looks the same as the input text. Thus if there is bad spelling in the original (for example inappropriate multiple vowels), this code does not fix that or mark an error, it simply passes it on for other processes to handle appropriately. For brevity, only formal confusables are resolved. Informal confusables are left as an extra step for those wanting to formalise them. Without informal confusable expansion, the output string is never longer than the input string.
The reference implementation is in python3 using only core modules and is written to be easily translatable into other languages. The code is not written for speed, but for clarity.

```
#!/usr/bin/python3
#  Copyright (c) 2021, SIL International.
#  Licensed under MIT license: https://opensource.org/licenses/MIT

import enum, re

class Cats(enum.Enum):
    Other = 0; Base = 1; Robat = 2; Coeng = 3; Z = 4
    Shift = 5; Vowel = 6; MS = 7; MF = 8

categories =  ([Cats.Base] * 52       # 1780-17B3
             + [Cats.Vowel] * 18      # 17B4-17C5
             + [Cats.MS]              # 17C6
             + [Cats.MF] * 2          # 17C7-17C8
             + [Cats.Shift] * 2       # 17C9-17CA
             + [Cats.MS]              # 17CB
             + [Cats.Robat]           # 17CC
             + [Cats.MS] * 5          # 17CD-17D1
             + [Cats.Coeng]           # 17D2
             + [Cats.MS]              # 17D3
             + [Cats.Other] * 9       # 17D4-17DC
             + [Cats.MS])             # 17DD

khres = {    #  useful regular sub expressions used later
    "BNB":   "[\u1780-\u1793\u1795-\u17A2]",
    "SF":    "[\u179E-\u17A0\u17A2]",
    "SNF":   "[\u1780-\u179D\u17A1]",
    "SS":   "[\u1784\u1789\u1793\u1794\u1798-\u179D]",
    "VA":   "[\u17B7-\u17BA\u17BE\u17D0\u17DD]|\u17B6\u17C6",
}

def charcat(c):
    ''' Returns the Khmer character category for a single char string'''
    o = ord(c)
    if 0x1780 <= o <= 0x17DD:
        return categories[o-0x1780]
    elif o in (0x200C, 0x200D):
        return Cats.Z
    return Cats.Other

def khnormal(txt):
    ''' Returns khmer normalised string, without fixing or marking errors'''
    # categorise every character in the string
    charcats = [charcat(c) for c in txt]

    # Recategorise base → coeng after coeng char
    for i in range(len(charcats)-1, 0, -1):
        if charcats[i-1] == Cats.Coeng and charcats[i] == Cats.Base:
            charcats[i] = Cats.Coeng

    # find subranges of base+non other and sort components in the subrange
    i = 0
    res = []
    while i < len(charcats):
        c = charcats[i]
        if c != Cats.Base:
            res.append(txt[i])
            i += 1
            continue
        # scan for end of syllable
        j = i + 1
        while j < len(charcats) and charcats[j].value > Cats.Base.value:
            j += 1
        # sort syllable based on character categories
        # sort the char indices by category then position in string
        newindices = sorted(range(i, j), key=lambda e:(charcats[e].value, e))
        replaces = "".join(txt[n] for n in newindices)

        replaces = re.sub("([\u200C\u200D])[\u200C\u200D]+", r"\1", replaces) # remove multiple ZW(N)J
        replaces = re.sub("\u17C1(\u17BB?)\u17B8", "\\1\u17BE", replaces) # compose split vowels
        replaces = re.sub("\u17C1(\u17BB?)\u17B6", "\\1\u17C4", replaces)
        replaces = re.sub("\u17B8(\u17BB?)\u17C1", "\\1\u17BE", replaces)
        replaces = re.sub("\u17B6(\u17BB?)\u17C1", "\\1\u17C4", replaces)
        replaces = re.sub("({VA})(\u17BB)".format(**khres), r"\2\1", replaces) # reorder u before VA
        replaces = re.sub("({SF}(?:\u17D2{BNB}){{0,2}}|{BNB}(?:\u17D2{SF}(?:\u17D2{BNB})?"
                          "|\u17D2{BNB}\u17D2{SF}))\u17BB({VA})".format(**khres),
                             "\\1\u17CA\\2", replaces)              # Upshifting triisap
        replaces = re.sub("({SS}(?:\u17D2{SNF}){{0,2}}|{SNF}(?:\u17D2{SS}(?:\u17D2{SNF})?"
                         "|\u17D2{SNF}\u17D2{SS}))\u17BB({VA})".format(**khres),
                             "\\1\u17C9\\2", replaces)              # Upshifting muusikatoan
        replaces = re.sub("(\u17D2\u179A)(\u17D2[\u1780-\u17B3])", r"\2\1", replaces) # coeng ro 2nd
        replaces = re.sub("(\u17D2)\u178A", "\\1\u178F", replaces)  # coeng da → ta

        res.append(replaces)
        i = j
    return "".join(res)

if __name__ == "__main__":
    import sys
```

```
if len(sys.argv) < 2:
     print("khnormal infile [outfile]")
     sys.exit(1)
infile = open(sys.argv[1], encoding="utf-8")
outfile = open(sys.argv[2], "w", encoding="utf-8") if len(sys.argv) > 2 else sys.stdout
for l in infile.readlines():
     outfile.write(khnormal(l))
```

Shaping and Font Development

The only real concern of the font is that there be a visual distinction between any two different strings. While the regular expression describes what constitutes a 'legal' sequence, it gives no indication of how illegal sequences are to be indicated. Where should a dotted circle be inserted? This section does not distinguish responsibilities between the shaper and the font. In effect it says that ensuring good rendering, including marking of errors, is the responsibility of the font, and behind that any shaping engine that helps the font do its work. The reason for this is that at the time of writing, shaping support for Khmer can be very different across different shapers and it is unknown what shaping support will be available to fonts in the future

An important consideration is the difference in shaping needs between Modern Khmer and Middle Khmer. Middle Khmer allows such things as multiple vowels and final coengs. These are problematic in Modern Khmer. There are various options in how to deal with the contrast:

1. Unify the syllable descriptions and have one shaper for both orthography families.
2. Distinguish syllable structures of Modern and Middle Khmer and allow vowel sequences and final coengs only in Middle Khmer.

These each have costs and benefits.
Option 1 is simpler to implement. But it comes with a huge cost and that is that font designers have to ensure and test their fonts for Middle Khmer as well as Modern Khmer, even if they have no interest in supporting Middle Khmer. In addition, while it is anticipated that most of the heavy lifting of hiding the encoding complexities from users will be done by the keyboard implementation, it is probable that there will be simple keyboards produced that do not do the necessary work and will push the cognitive load on users. This is not good, but is made much worse if they do not constrain users from typing sequences that are not legal for Modern Khmer but are for Middle Khmer and yet are assumed to be in Modern Khmer. It would place an immense burden on font developers of Modern Khmer fonts to handle the erroneous sequences in the font rather than having the necessary shaper support..
Option 2 uses some mechanism to distinguish Modern and Middle Khmer. This cannot be done by analysis of the text data itself precisely because a Modern Khmer shaper has to mark strings that a Middle Khmer shaper would accept as illegal. So saying: 'aha this string has Middle Khmer type structures in it', does nobody any service to identify the text as Middle Khmer when it should be marked as illegal Modern Khmer. The current approach is to use language marking of text. Thus text would either be identified as Modern Khmer (and if unmarked) or as Middle Khmer. This places a burden on the far smaller Middle Khmer user community to mark their text for language.
The issue is that the concept of changing shapers (or such radical shaping behaviour as the grapheme cluster analysis) based on language rather than script alone, is novel and will require engineering support. Another advantage of option 2 is that it allows a font designer only interested in supporting Modern Khmer to indicate that in their font and for the shaper to ensure that all data the font renders is interpreted as Modern Khmer.
To enable this option, a suitable language tag would need to be agreed for Middle Khmer and also a mapping to an OpenType language tag. Notice that the only ISO639 codes for Khmer are: kh for Modern Khmer, kxm for Northern Khmer (a minority language that can use Khmer script) and okz for Old Khmer. Middle Khmer is not considered a different language to Modern Khmer, although its orthography is very different. This would probably result in a language tag with a variant against a different ISO639 code.

Shaping

The description here does not attempt to use any existing shaping specification. Instead it outlines the needs of a shaper without specifying the actual implementation.

The primary issue in creating a font is to ensure that no two different strings look the same. This is a shared responsibility between the shaping engine and the font. There is a tendency to not want to overburden the shaping engine with a lot of detailed rules, but equally to provide helpful support to the font.

The full disambiguating regular expression is large and complex. But it is unlikely that the regular expression will be implemented directly, and would be expressed in code in a different way. This is especially true since most regular expression engines cannot handle variable length zero width look behind assertions. That is one way of identifying ‘illegal’ strings. But another way allows for a more nuanced insertion of error marks. This approach is to do a very basic syllable analysis and then use negative rules to identify and mark bad strings.

Consonant Shifters

As an example, let’s consider the consonant shifter. The regular expression for this particular part of the string is certainly complex. How might a font (in conjunction with a shaper) handle this? It already has to address such sequences to identify when the shifter downshifts, so perhaps we can do all of the work of downshifting and error marking together. The examples below will use the FEA syntax and presumes no shaper support.

For the sake of this example, we will identify all base forms by their unicode value with glyph names such as `u179A`. Notice that at this point we are dealing with glyphs and not codepoints. In addition, every consonant has a coeng form: the glyph that is used when the consonant is preceded by a coeng (`17D2`) and this is named as a variant glyph: thus `u179A.coeng`. We also presume that each of the classes in the regular expression has a corresponding class of glyphs. If the class is preceded by a coeng (`17D2`), then the class is simply a class of `.coeng` forms. Thus:

```
@SCS = (u1784.coeng u1789.coeng u1799.coeng u179A.coeng u179C.coeng u179D.coeng)
@SS = (u1784 u1789 u1798 u1799 u179A u179C u179D)
```

The task of the lookup we are to write is to process `u17C9` and `u17CA`. There are two actions we can take: we can convert the shifter into a -u vowel (`u17BB`) or insert a dotted circle before it (`u25CC`). There is also a lookup to strip the zwj or zwnj. In the case of zwnj, if it is legal, then it is downshifting and we replace it with a -u glyph, whereas with zwj we leave the consonant shifter in place.

```
lookup shiftu {
    sub u17C9 by u17BB;
    sub u17CA by u17BB;
} shiftu;

lookup shifterr {
    sub u17C9 by u25CC u17C9;
    sub u17CA by u25CC u17CA;
} shifterr;

lookup shiftstrip {
    sub u200D u17C9 by u17C9;
    sub u200D u17CA by u17CA;
    sub u200C u17C9 by u17BB;
    sub u200C u17CA by u17BB;
} shiftstrip;
```

These lookups are ‘called’ from a contextual chaining lookup that uses strings of glyphs to call the appropriate lookup on the shifter. Because of the size of this lookup here, we only consider u17CA. Handling u17C9 is left as an exercise for the reader. We also leave out the u17B6 u17C6 vowel context after the first few example rules.

```
lookup doshift {
    # unmarked downshift
    sub @SF u17CA’ lookup shiftu @VS;
    sub @SF u17CA’ lookup shiftu u17B6 u17C6;
    sub @B @SCF u17CA’ lookup shiftu @VS;
    sub @B @SCF u17CA’ lookup shiftu u17B6 u17C6;
    sub @B @SCF @C u17CA’ lookup shiftu @VS;
    sub @B @SCNF @SCF u17CA’ lookup shiftu @VS;
    # ZWNJ stops downshift
    sub @SF u200C’ lookup shiftstrip u17CA’ @VS;
    sub @B @SCF u200C’ lookup shiftstrip u17CA’ @VS;
    sub @B @SCF @C u200C’ lookup shiftstrip u17CA’ @VS;
    sub @B @SCNF @SCF u200C’ lookup shiftstrip u17CA’ @VS;
    # ZWJ should not occur
    sub @SF u200D’ lookup shifterr u17CA’ @VS;
    sub @B @SCF u200D’ lookup shifterr u17CA’ @VS;
    sub @B @SCF @C u200D’ lookup shifterr u17CA’ @VS;
    sub @B @SCNF @SCF u200D’ lookup shifterr u17CA’ @VS;

    # we only get here if all the other rules fail, so our classes can be vague.
    # explicit ZWJ downshifts
    sub @B u200D’ lookup shiftu u17CA’ @VS;
    sub @B @C u200D’ lookup shiftu u17CA’ @VS;
    sub @B @C @C u200D’ lookup shiftu u17CA’ @VS;
    # explicit ZWNJ should not occur
    sub @B u200C’ lookup shifterr u17CA’ @VS;
    sub @B @C u200C’ lookup shifterr u17CA’ @VS;
    sub @B @C @C u200C’ lookup shifterr u17CA’ @VS;
} doshift;
```

Tests

- Ensure coeng sequences are contrastive based on order (reversing the order should result in a different rendering)
- Coeng ta and coeng da should render contrastively.
- If a coeng ro is stored first in a coeng sequence, the font should show a visual contrast (however ugly that may be) compared to the coeng ro coming second.
- If a consonant shifter precedes a spacing coeng, the font (or ideally the shaper) should show a visual contrast compared to when the consonant shifter is after the coeng.

Design Issues

Coeng Stacking

The stacking of below diacritics is a particular design issue for Khmer fonts. Long stacks of say two coengs and a lower diacritic vowel do exist, but they are rare. A typesetter does not want to allocate lots of space below the baseline just for the rare cases that may occur once or twice in a book. Therefore there is a desire by font designers to want to reduce the height of the lower diacritic stack. For example, stacking lower diacritics horizontally across the bottom of the base character. But this flies in the face of readability where readers need to know the order of the coengs in order to read well. In addition, fonts need to show coeng order so that they show a visual distinction for a different character sequence.
If a font designer wants to design with horizontal stacking, they need to ensure that reversing the order of the coengs in the stack leads to a different visual representation. Such stacks occur with borrowed words. Thus ‘Florida’: ហ្វ្លរីដា[7]? ហ្ល្វរីដា? ផ្លរីដា? Or ‘free’: ហ្វ្រី, or ‘three’: ស្រ៊ី. Another good example is ទទ្ឡ្ហីករណ៍.

Some other examples of the problems of combining coengs show the problems that users face on various

[7] Note that this document was written in Google docs and there are no known fonts available in Google Fonts, that render these words correctly.

platforms.

ក្នុង ទ្ឃី ហ្ឈូ ផ្នុ Noto Sans Khmer UI (Android)

ក្នុង ទ្ឃី ហ្ឈូ ផ្នុ Leelawadee UI (Windows)

Independent Vowels and Coengs

The definition of bases includes independent vowels. This means that independent vowels should be able to take coengs. But some of the independent vowels have below baseline components which are problematic when combining with coengs. But there are no known cases of such sequences occurring in any use of the Khmer script. Thus font designers are free to make such sequences render poorly, either through collision or inserting a dotted circle, for example. The only independent vowel and coeng sequences which occur are [17B1 17B2] 17D2 1799. This problem also applies to 17A1 although there are a rare examples of coengs applying to 17A1: ទឡ្ហីករណ៍. It is unknown whether Middle Khmer has independent vowels with coengs.

Acknowledgements

The research and development of this document was done because a number of technical experts saw key technical problems with the Khmer encoding. This document is written to provide input both to the Unicode Technical Committee and also to the Cambodian Government. Thanks are due to SIL International and the National Polytechnic Institute of Cambodia Language Software Development Unit (NPIC LSDU) for their authorship. Martin Hosken's research is also undertaken under the auspices of the Payap University Linguistics Institute, ChiangMai Thailand. Most of the underlying research was undertaken in conjunction with Makara Sok and Didi Kanjahn and based on the encoding proposed by Javier Solá (Open Institute). Thanks are also due to the following people for their technical contributions and review: Peter Constable, Sovichet Tep, Nathan Willis and in particular Norbert Lindenberg who generously gave thorough testing. Trent Walker (Stanford University) reviewed and contributed to the section on Middler Khmer.

Appendix 1 - Current Confusion

The problem that the Khmer script faces is that there is often more than one way to encode a visual string and it be valid, and yet only one way is the ‘correct’ way. For those very experienced in understanding how the Khmer script is stored in Unicode, and with a strong linguistic awareness, there seems to be no problem. But for many users there are problems. Here we present some simple statistics taken from page counts in Google for different encodings of the same word.

Example Words

‘Woman’

The word ស្ត្រី `woman` consists of 4 characters. The initial consonant is not in confusion. The coeng ta is confusible with coeng da and the 3 characters following the base consonant may occur in any order. The result is 12 possible ways of encoding the word. They are listed here along with the number of pages Google found of that spelling, in popularity order. Notice that the most popular spelling is not the ‘correct’ spelling. The best that someone searching for this word can hope for is 53% by typing their search term wrongly.

Text	Romanised	Popularity
ស្រ្តី	srti	8,950,000
ស្ត្រី	stri	4,950,000 (correct spelling)
ស្រ្ដី	srdi	1,340,000
ស្ដ្រី	sdri	893,000
ស្រី្ត	srit	620,000
ស្តី្រ	stir	25,900
ស្រី្ដ	srid	19,600
ស្ដី្រ	sdir	6,190
សី្ត្រ	sitr	10
សី្ដ្រ	sidr	3
សី្រ្ត	sirt	1
សី្រ្ដ	sird	1

‘Detect/investigate’

ស៊ើប ‘to detect/investigate’ consists of a confusable vowel ើ (17BE) and a consonant shifter (៊ 17CA) which is down shifted to be a glyph which looks like that of 17BB in this context. There are 15 possible ways of typing this word. The two most noticeable errors are (1) 17BE is perceived as a combination of two separate vowels, i.e. 17C1 and 17B8 and (2) 17CA is thought to be 17BB because of how it looks.

Text	Sequences	Popularity
ស៊ើប	ស៊ ើ ប	759,000 (correct spelling)
សើុប	សើ ុ ប	38,500
ស៊េីប	ស៊ េ ី ប	11,800
ស់ើប	ស់ ើ ប	11,600 *
សេីុប	ស េី ុ ប	6,410

ស៊ើប	សុ ី ប	5,120
ស៊ើប	ស ើុ ប	30
ស៊ែីប	សុ ែី ប	21
ស៊ីេប	ស៊ី េ ប	14
សើ៊ប	សី៊ ប	11 *
សុីេប	សុ ី េ ប	8
សើ់ប	ស់ ើ ប	8 *
ស់ីេប	ស់ ី េ ប	6 *
សើុប	សី េុ ប	5
សីុេប	សី ុ េ ប	3

Entries marked with * are not automatically fixed by the reference normalization code.

‘Eat’

Here, there is considerable downshifting confusion.

Text	Sequences	Popularity
ស៊ី	ស៊ី	3,250,000 (correct spelling)
សុី	សុី	3,040,000
ស់ី	ស់ី	632,000 *
សីុ	សីុ	400,000
សី៊	សី៊	364
សី់	សី់	9 *

Entries marked with * are not automatically fixed by the reference normalization code.

‘Bread’

More downshifting confusion with a less common word.

Text	Sequences	Popularity
ប៉័ង	បុ ័ ង	34,700
ប៉័ង	ប៉ ័ ង	26,400 (correct spelling)
ប៉័ង	ប័ ុ ង	16,700

‘One sort’

For the most part people realise that the downshifter goes after the coeng.

Text	Sequences	Popularity
ម្យ៉ាង	ម ្យ ៉ ា ង	2,530,000 (correct spelling)
ម៉្យាង	ម៉ ្យ ា ង	480,000

‘Don’t’

People tend not to have a problem with typing a vowel before a final.

Text	Sequences	Popularity

កុំ	កំុ	6,050,000 (correct spelling)
កុំ	កំ ុ	284,000

‘Wait’

People tend not to fall into the trap of typing ‘am’ the Thai way.

Text	Sequences	Popularity
ចាំ	ច ាំ	5,860,000 (correct spelling)
ចាំ	ចំ ា	208,000

‘Mr’

Most people only use one vowel for split vowels.

Text	Sequences	Popularity
លោក	លា ក	8,190,000 (correct spelling)
លោក	ល េា ក	422,000
លាេក	លា េ ក	15,600

‘Khmer’

People do well at spelling the language name correctly.

Text	Sequences	Popularity
ខ្មែរ	ខ្ម ែ រ	29,100,000 (correct spelling)
ខ្មែរ	ខ ្មែ រ	372,000

Coeng Ta vs Coeng Da

Here we see some example words and their various popularities for the confusion between coeng ta and coeng da. In each case the ‘correct’ spelling is emboldened.

ដ spelling	ដ popularity	ត spelling	ត popularity
កណ្ដាល	**2,120,000**	កណ្តាល	3,550,000
ក្ដី	**1,660,000**	ក្តី	3,560,000
ស្ដាំ	**445,000**	ស្តាំ	903,000

While students are taught the correct usage of coeng ta and coeng da, it does not mean that the lessons are always remembered into adulthood.

Bibliography

Antelme, Michel, 2007 "INVENTAIRE PROVISOIRE DES CARACTÈRES ET DIVERS SIGNES DES ÉCRITURES KHMÈRES PRÉ-MODERNES ET MODERNES EMPLOYÉS POUR LA NOTATION DU KHMER, DU SIAMOIS, DES DIALECTES THAÏS MÉRIDIONAUX, DU SANSKRIT ET DU PĀLI ∗" (Projet "Corpus des inscriptions khmères" —CIK), url: https://web.archive.org/web/20190814131541/http://aefek.free.fr/iso_album/antelme_bis.pdf

Bernard, J. B., 1902 "Dictionnaire Cambodgien-Francais" (HongKong)

British and Foreign Bible Society, 1899 "The Gospel According to Luke", url: https://books.google.com.kh/books?id=ZrQUAAAAYAAJ

Chuon Nath, 1967 "Khmer Dictionary" (Buddhist Institute 5th ed, 1967/68), url: windows version http://krou.moeys.gov.kh/kh/article/item/1135-%E1%9E%9C%E1%9E%85%E1%9E%93%E1%9E%B6%E1%9E%93%E1%9E%BB%E1%9E%80%E1%9F%92%E1%9E%9A%E1%9E%98%E2%80%8B%E1%9E%81%E1%9F%92%E1%9E%98%E1%9F%82%E1%9E%9A.html#.YLmnD_kvPIU, online version http://dictionary.tovnah.com/help

Sok, Makara, 2016 "Phonological Principles and Automatic Phonemic and Phonetic Transcription of Khmer Words" (Payap University, MA Thesis, 2016), url: https://drive.google.com/file/d/1c_FXNy90pv06StsBMQz4Rzk87ulMqXyM/view?usp=sharing

Lindenberg, Norbert, 2019: Issues in Khmer syllable validation. (Lindenberg Software, 2019), url: https://lindenbergsoftware.com/en/notes/issues-in-khmer-syllable-validation/

Solá, Javier, 2004 "Issues in Khmer Unicode 4.0" (Open Forum of Cambodia, version 2.0, 21/Oct/2004), url: https://sourceforge.net/projects/khmer/files/Documents%20about%20Khmer%20script/Documents%20about%20Khmer%20Script%20and%20about%20Khmer%20Unicode%20v1.0/IssuesInUnicode40-v2.0.pdf/download

Valy, D., Verleysen, M., Chhun, S., & Burie, J. C., 2017 "A New Khmer Palm Leaf Manuscript Dataset for Document Analysis and Recognition - SleukRith Set" (In 4th International Workshop on Historical Document Imaging and Processing (HIP), DOI 10.1145/3151509.3151510, Data: https://github.com/donavaly/SleukRith-Set).