Note: This document is intended for discussion only.

L2/19-365

Structuring Tai Tham Unicode

M. Hosken, SIL International, Payap University
Revision 7

Table of Contents

1 Introduction

This paper is a draft and is liable to change significantly. As such it also falls fowl of: “Why did you not talk about *x*?” Probably because I didn’t think of it; or “Why did you not describe *y* more clearly?” Probably because my brain isn’t the same as yours. In either case I am happy to try to improve things to make things clearer and more accurate. But my primary concern is that we have a good encoding and not necessarily a perfect paper describing it.

The Tai Tham script is one of the more complex scripts in Unicode. In many respects it behaves like the Myanmar script, but with added complexity. It uses subjoined consonants not just for syllable chaining but for initial clusters, ala Devanagari, for final consonants ala Old Khmer, and even for vowels. In addition, Tai Tham based orthographies typically have no word spaces[1] and, missing any phonetic syllable separators, therefore, has the added ambiguity of whether a base consonant is a final or initial. As a result, Tai Tham has some inherent visual ambiguities. Having said that, people are still able to read, and there are a number of reading principles that, certainly for modern spelling conventions, allow readers to minimise the ambiguity. The problem is that these principles are very hard to codify and are too complex for the generic computing models into which the script may be desired to fit.

The upshot of all this is that when the script was added to the standard, a relatively laissez faire attitude was taken to encoding order. The expectation was that users would type and store data in reading order. In effect, the reading ambiguities would be removed in stored text. The reason for this is that there has been considerable variation in spelling and types of syllable structure in how people have written using the Tai Tham script, and if the addition of Tai Tham to Unicode had had to wait until a full analysis had been done, Tai Tham would still not be in Unicode. But the resulting encoding order approach is problematic since it results in over differentiation. Two different, but correctly encoded, strings may render identically and there is no way to distinguish them. Tai Tham is unique in Unicode in the amount of over differentiation that it supports.

If over differentiation is bad for encodings, under differentiation is even worse. Under differentiation is where it is not possible to encode a necessary visual difference and is considered a technical deficiency of an encoding that is sufficient justification to change an encoding in Unicode. It is where two different visual representations require the same underlying string (and therefore one of the representations is lost, or out of band information is necessary to resolve the rendering ambiguity). A good encoding design, therefore, requires that it not contain over differentiation or under differentiation. For Tai Tham, due to its visual complexity, this is very hard.

Another aspect to encoding is whether an encoding is primarily visual or logical. Does it store the surface rendering form or the underlying reading form. The current Unicode standard of Tai Tham is entirely a logical encoding. But this leads to huge over differentiation. The problem of going to an entirely visual encoding is that it makes analysis really hard. For example, culturally appropriate collation becomes nearly impossible. Tai Tham collation is particularly problematic since most sort orders require the user to be able to read the word and to use the phonemic representation for collation. This goes against all the principles of computer collation. The complexity of converting from a visual form to a logical one is huge and beyond what most sorting systems can handle. The question this paper examines is whether there is some compromise position for an encoding that is sufficiently logical to allow analysis while also straightforward to render.

A further dimension to the question comes from input. The current encoding of Tai Tham requires that a typist be able to read what they type correctly. They cannot merely look at a text and type it. They have to, in effect, read it and then type the logical representation of that reading. Along with this, they have to get all the diacritics in the right relative order. If they get it wrong, at best they may see what looks like a perfectly good word being marked as a spelling error. But more often, they just end up with inconsistently encoded text which makes searching and analysis nearly impossible.

1 Modern Pali being an exception.

The purpose of this paper is to propose a new encoding order for Tai Tham that uses all the same codepoints in Unicode but that reduces over differentiation, preferably removing it altogether. In addition, it aims to be one that lends itself to analysis by, for example keeping vowel sequences together as much as possible and storing tone after vowels, etc. Obviously, the encoding aims to not be under differentiated with respect to the needs of its user community. Tai Tham script usage is poised for much needed deployment and wide usage. Currently there is relatively little data stored and so there is a window of opportunity to make the changes needed to provide a good basis for future computing in the script.

The paper starts with a description of the current encoding and then moves into proposing what changes can be made to reduce the structural complexity of the encoding such that it can be rendered and keyed. From here we examine implementation details, particularly with regard to the Universal Shaping Engine for OpenType support and how to normalize text into a canonical form of particular relevance to keyboard entry.

A number of examples are given in tables throughout the paper. Each example shows a Lanna style rendering, a Khuen style rendering and then the underlying stored codes. Two renderings are shown since they can be quite different.

2 Syllable Structure

In this section we look at the structure of an orthographic syllable. This may well differ from a linguistic syllable in that it is primarily a graphical construct, that while motivated by the phonemics it conveys, is not governed by them. Tai Tham loosely follows the abugida structure of syllables, but stretches it beyond what many may consider the bounds of good order! The description here is introductory and more thorough treatments may be found in the literature. In the various subsections of this section we build up a regular expression description of the orthographic syllable[2].

The basic structure we examine is:

S = C M V T F

C – Consonant and required; M – Medial; V – Vowel; T – Tone; F – Final. Only C is required and any of the components may consist of a sequence of Unicode characters. This is not the only structure as we will see.

2.1 Consonants

A consonant consists of any character in the range U+1A20 – U+1A4C. There are also independent vowels that may take this initial position in the range U+1A4D – U+1A52. There are two ligatures that may also start an orthographic syllable: U+1A53 (ᩓ LAE), U+1A54 (ᩔ GREAT SA).

One situation in which the initial consonant is a sequence is when considering the tone class prefix h (U+1A49) as in this table which shows a Lanna style rendering in the first column, a Khuen style rendering in the second and the sequence itself in the third.

ᩉᩣᨶ	ᩉᩣᨾ	1A49 (ᩉ HA) 1A60 SAKOT 1A36 (ᩣ NA) 1A62 (ᩮ MAI SAT) 1A26 (ᨶ NGA)

Thus in regular expression terms we can say:

C = [\u1A20-\u1A54] | \u1A49 \u1A60 [\u1A20-\u1A4A]

2.2 Medials

The most common expression of a medial is the use one of the medial characters: U+1A55 (ᩕ MEDIAL RA), U+1A56 (ᩖ MEDIAL LA), medial ya encoded as U+1A60 SAKOT U+1A3F (ᩡ YA), and medial wa encoded as U+1A60 SAKOT U+1A45 (ᩢ WA). Notice the number of places that this sequence crops up as a valid sequence.

There is some ambiguity over the use of medial la, U+1A56. Some fonts do not contrast the visual presentation of the medial la over subjoined la (U+1A60 U+1A43). This is unfortunate since it is important that the distinction is made. If there is no contrast in a font, then someone may not type the contrast and then if the text is rendered in a font with the contrast, the resulting text may be wrong. In addition, it means the typist may be introducing spelling errors that they cannot see in the font with no contrast. Notice that in the h prefix context, the medial la is used:

ᩉ᩠ᩃ	ᩋ	1A49 (ᩉ HA) 1A60 SAKOT 1A43 (ᩃ LA)
ᩉ᩠ᩈ	ᩌ	1A49 (ᩉ HA) 1A56 (ᩖ MEDIAL LA)
ᩉ᩠ᩈᨾ	ᩌᨾ	1A49 (ᩉ HA) 1A56 (ᩖ MEDIAL LA) 1A62 (ᩮ MAI SAT) 1A26 (ᨶ NGA)
ᨠᩡ	ᩍᩴ	1A23 (ᨠ KA) 1A60 SAKOT 1A3F (ᩡ YA) 1A69 (ᩩ U) 1A76 (᩶ TONE-2)
ᨠᩢᨿ	ᩍᨿ	1A23 (ᨠ KA) 1A60 SAKOT 1A45 (ᩢ WA) 1A63 (ᩣ AA) 1A60 SAKOT 1A3F (ᩡ YA)

2 For those turning this into a full regular expression, there is an implied parenthesis around each defined subexpression and spaces may be removed.

Medial ya and medial wa may also act as vowels. There is a visual ambiguity here. Rather than carry that ambiguity into the regular expression, we allow for a miscategorisation of a vowel as a medial, and then allow multiple medials. This is not a problem since the primary concern with such string processing is ensuring a consistent order rather than precise categorisation of components. There is only visual confusion over the relative order of U+1A55 (◌᩵ MEDIAL RA). The vertical stacking of medials/vowels is achieved given the order listed. Removing the medials from what may occur subjoined after an h, allows us to simplify and reduce the list.

C = [\u1A20-\u1A54] | \u1A49 \u1A60 [\u1A20-\u1A3E]
M = \u1A55? \u1A56? (\u1A60\u1A45)? (\u1A60\u1A3F)?

2.3 Vowels

Unlike most Indic scripts, Tai Tham is used for languages that use much richer sequences of vowel characters. For consistent ordering, this means agreeing how compound vowel sequences are to be stored. The general principle is that vowels are stored as:

$V = V_p? V_b? V_a?$

While it is clear that V_p must come first, the other two are somewhat arbitrarily ordered. We follow the order in the original proposal, here.

We will discuss spacing vowels later. All vowel classes are optional.

V_p = [\u1A6E-\u1A72]
V_b = [\u1A69-\u1A6A\u1A6C]
V_a = [\u1A65-\u1A68\u1A6B\u1A73] [\u1A62\u1A74]?

Allowing U+1A62 (◌᩵ MAI SAT) in conjunction with other upper vowels is for the case where it acts as a final -k. U+1A74 (◌᩵ MAI KANG) has all kinds of functions and may combine with any vowel, but occurs before a tone mark in this context. Some special case sequences need consideration, in particular in relation to the medials that can also work as vowels:

ᨦᩮᩢ	ᨦᩮᩢ	1A26 (ᨦ NGA) 1A60 SAKOT 1A45 (◌ᩢ WA) 1A6B (◌ᩮ O) 1A75 (◌᩵ TONE-1)
ᨠᩣ	ᨠᩣ	1A20 (ᨠ KA) 1A60 SAKOT 1A45 (◌ᩣ WA) 1A76 (◌᩶ TONE-2) 1A60 SAKOT 1A3F (◌᩿ YA)

The tone marks help in identifying whether something is a medial, vowel or final, but even with these, the order is not necessarily cut and dried. Is the U+1A60 SAKOT U+1A45 (◌ᩢ WA) here acting as a medial or a vowel? To a large extent, it is not important since we are only interested in order and not categorisation.

2.4 Tones

Tone marks are stored after the vowel sequence. There can only ever be one if present. The only complexity with tones lies in their interaction with spacing vowels, to which we will return.

T = [\u1A75-\u1A79]

2.5 Finals

There are two kinds of finals: spacing and diacritical. Spacing finals are consonants that may in turn be modified through syllable chaining. Diacritical consonants are primarily subjoined consonants or a U+1A58 (◌᩷ MAI KANG LAI).

F_s = [\u1A4B\u1A54] | \u1A60? [\u1A20-\u1A4A]
F = F_s? \u1A58?

The restricted list of consonants is used because certain consonants are never used for finals. For example, letter A U+1A4B (ᩋ A) is used as the spacing component in a compound vowel and not as a final consonant. This list could be further constrained, but for little value.

2.6 Spacing Vowels

In addition to sequences of non spacing vowels, there may are also be spacing vowels. These can occur as part of a vowel sequence, but the overall sequence my be interrupted by a tone mark. Thus an alternative syllable pattern is:

C M V T V_s F

U+1A63 (ᩣ AA) is a common spacing vowel. U+1A64 (ᩤ TALL AA) is a variant form of the same vowel for use where there may be confusion over whether U+1A63 (ᩣ A) is a separate vowel or part of the preceding consonant, visually, to create another consonant and so cause reading confusion. U+1A61 (ᩡ A) is a short -a that can be used to shorten a vowel while also acting as a final consonant. U+1A4B (ᩋ A) can combine in a vowel sequence to be part of a complex vowel. Thus we get complex vowels like:

ᨠᩮᩬᩥ᩵ᩋᩡ	ᨠᩮᩬᩥ᩵ᩋᩡ	1A20 (ᨠ KA) 1A6E (◌ᩮ E) 1A6C (◌ᩬ OA) 1A65 (◌ᩥ I) 1A75 (◌᩵ TONE-1) 1A4B (ᩋ A) 1A61 (ᩡ A)
ᨠᩮᩣᩡ	ᨠᩮᩣᩡ	1A20 (ᨠ KA) 1A6E (◌ᩮ E) 1A63 (ᩣ AA) 1A61 (ᩡ A)

V_s = [\u1A63\u1A64] \u1A74? | [\u1A4B\u1A63]? \u1A61

The U+1A4B (ᩋ A) and U+1A61 (ᩡ A), in this context, are never followed by a final. U+1A4B (ᩋ A) may occur in a syllable chaining context (see below). One refactoring is to integrate U+1A61 (ᩡ A) into F_s and simplify the regular expression. This then allows for finals following a spacing vowel. For example:

ᨻ᩠ᨿᩣ᩠ᨲᩥ	ᨻ᩠ᨿᩣ᩠ᨲᩥ	1A3B (ᨻ PA) 1A60 SAKOT 1A3F (◌ᩕ YA) 1A63 (ᩣ AA) 1a60 SAKOT 1a35 (◌᩠ᨲ TA) 1A65 (◌ᩥ I)

Sara am is encoded U+1A63 (ᩣ A) U+1A74 (◌ᩴ MAI KANG) and takes no following subscript consonants.

As a result, the regular expression is restructured to accentuate the cluster structure of a syllable:

S = CC (V_s | V_f)?
CC = C M? V? T? F?
C = [\u1A20-\u1A54] [\u1A5A\u1A5B]? |
  \u1A49 \u1A60 [\u1A20-\u1A3E]
V_s = AV F?
AV = [\u1A63\u1A64] \u1A74?
V_f = \u1A4B? \u1A61

Here we are saying a syllable consists of a consonant cluster (CC) followed optionally by either a spacing vowel (that can take a final) or a vowel that is itself a final. The CC is a single cluster separated from any following clusters, except for the case of sara am where the sequence U+1A63/4 (ᩣ AA) U+1A74 (◌ᩴ MAI KANG) has the U+1A74 (◌ᩴ MAI KANG) rendered as part of the preceding CC cluster, in Lanna. There are other visual ligation and rendering that may also occur, for example na (ᨶᩣ) U+1A36 (ᨶ NA) U+1A63 (ᩣ AA).

ᨣ᩶ᩣᩴ	ᨣ᩶ᩣᩴ	1A23 (ᨣ KA) 1A76 (◌᩶ TONE-2) 1A63 (ᩣ AA) 1A74 (◌ᩴ MAI KANG)

2.7 Syllable Chaining

In addition to having subjoined finals, Tai Tham allows syllable chaining as found in the Myanmar script, along with others. Here a spacing final is followed by a subjoined consonant that is the start of the next syllable in the word. The result is some sequences that would otherwise be rarely seen.

ᨠᩮ᩠ᨲᩩ	ᨠᩮ᩠ᨲᩩ	1A20 (ᨠ KA) 1A6E (◌ᩮ E) 1A60 SAKOT 1A32 (◌᩠ᨲ TA) 1A69 (◌ᩩ U)
ᨶᩬᨾ᩠ᨾᩪᨶᩥᨿᩴ᩠ᨲ	ᨶᩬᨾ᩠ᨾᩪᨶᩥᨿᩴ᩠ᨲ	1A36 (ᨶ NA) 1A6C (◌ᩬ OA) 1A3E (ᨾ MA) 1A60 SAKOT 1A3E (◌᩠ᨾ MA) 1A6A (◌ᩪ UU) 1A36 (ᨶ NA) 1A65 (◌ᩥ I) 1A3F (ᨿ YA) 1A74 (◌ᩴ MAI KANG) 1A60 SAKOT 1A32 (◌᩠ᨲ TA)

The way to include syllable chaining is through recursion in the regular expression. As a result, though, the expression is no longer truly regular. This is addressed in a later section.

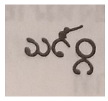
```
F = Fs? [\u1A58]? (\u1A60 S)?
```

Including syllable chaining in the definition of a syllable can result in what are linguistically polysyllabic syllables. For example, ᨵᨾ᩠ᨾᨬ᩠ᨬᩮᨷ᩠ᨷ. This is undesirable and a more refined definition of an orthographic syllable would address this issue. One solution is to break the chain by treating a spacing final as the start of a new syllable. This is not linguistically correct, but it does reduce the complexity of a sequence.

```
Fs = [\u1A4B\u1A54] | \u1A60 [\u1A20-\u1A4A]
```

At this point our orthographic cluster starts to look a lot like and extended grapheme cluster.

2.8 Mai Kang Lai



U+1A58 (◌ᩘ MAI KANG LAI) represents a syllable final -ng. It may be placed in different places according to font preference. It may occur over the top right of the cluster it follows (as in the Khuen example), or it may occur between the cluster and the next cluster or it may occur over the start of the next cluster, like a kinzi in Burmese. This can even extend to being over a prevowel, as shown here from a Northern Thai Dictionary (Udom 2004). But this could well be a misrender.

In Khuen the convention is to render the U+1A58 (◌ᩘ MAI KANG LAI) over the right hand side of the base. Which is somewhat in the space between the clusters. (Tengok).

The key thing here is that the position of U+1A58 (◌ᩘ MAI KANG LAI) is font specific.

2.9 Mai Sam

U+1A7B (◌᩻ MAI SAM) has a number of functions. One of these is to indicate that a following subjoined character is the start of a new syllable. In the original Unicode proposal, the mai sam is stored immediately after the new syllable initial. For example:

ᩈ᩠ᩃ᩻ᩪ	ᩈ᩠ᩃ᩻ᩪ	1A48 (ᩈ SA) 1A60 SAKOT 1A43 (◌᩠ᩃ LA) 1A7B (◌᩻ MAI SAM) 1A6A (◌ᩪ UU)

One way of adding this to our regular expression is through recursion as part of the initial consonant sequence:

```
C = [\u1A20-\u1A54] \u1A7B? | \u1A49 \u1A60 [\u1A20-\u1A3E]
```

It is noticeable that different fonts position U+1A7B (◌᩻ MAI SAM) in different positions depending on the presence of another upper diacritic. For example, it can occur before the diacritic or even at the end of the syllable: ᩈ᩠ᩃ᩻ᩪᨯ vs ᩈ᩠ᩃᩪᨯ᩻ in two editions of the same dictionary.

Another use of mai sam is as a reduplication mark. For this, the character is stored at the end of the syllable.

The final use of mai sam is in the place of a double acting consonant, as such it is a kind of final. Both of these cases are handled by:

```
F = Fs? [\u1A58\u1A7B]? (\u1A60 S)?
```

The net effect of both of these positions is that U+1A7B (◌᩻ MAI SAM) immediately follows the consonant.

2.10 Mai Kang

U+1A74 MAI KANG has a number of functions, reflecting the use of U+0E4D (◌ NIKAHIT) in Thai script.

1. Final nasal as used in ‘am’ U+1A63 (◌ AA) U+1A74 (◌ MAI KANG) for example ᨣᩣᩴ. Also as a final -ng if no vowel is present: ᩉᩫ U+1A49 (◌ HA) U+1A60 SAKOT U+1A36 (◌ NA) U+1A74 (◌ MAI KANG), ‘nang’.
2. In combination with U+1A6C (◌ OA) as an open final to the vowel as in ᩈᩮᩴ U+1A21 (◌ KHA) 1A6C (◌OA) 1A74 (◌ MAI KANG) U+1A75 (◌ TONE-1).
3. In Khuen it can act as a superjoined wa, in place of the medial: ᩁᩪ U+1A21 (◌ KHA) U+1A74 (◌ MAI KANG) U+1A60 SAKOT U+1A3F (◌ YA). We encode it as a vowel in this context.

Since, in Lanna, MAI KANG in ‘am’ is rendered over the consonant, MAI KANG is not used over a consonant before a U+1A63 AA vowel. In some older spellings, this does occur.

2.11 Ra Haam

U+1A7A (◌ RA HAAM) acts as a silencer and is placed over anything that is to be silenced, including misspellings. In effect it can be used to mark a final consonant by silencing the inherent vowel on the consonant. It is stored after everything else but before a U+1A7B (◌ MAI SAM):

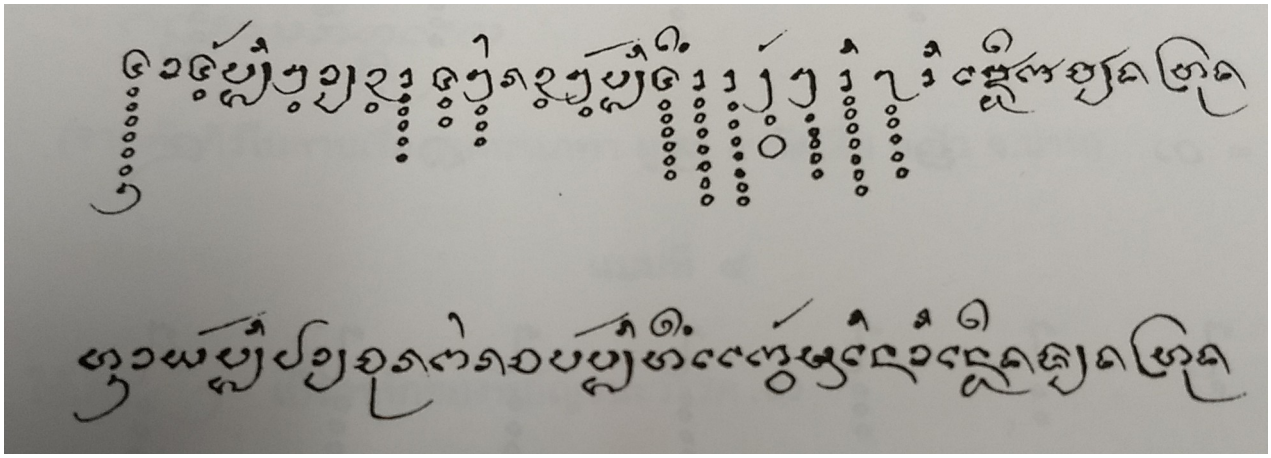
```
SM = [\u1A7A\u1A7C]? \u1A7B?
```

To support this, we add U+1A7B (◌ MAI SAM) to the syllable modifiers. Notice that U+1A7C KHUEN-LUE KARAN has the same function in Khuen.

There is some confusion over why there are separate codes for the same function in Lanna versus Khuen. Lanna uses U+1A58 MAI KANG LAI and U+1A7A RA HAAM, while Kuen uses U+1A58 MAI KANG LAI and U+1A7C KHUEN-LU KARAN. The reason that a common code couldn’t be used and styled appropriately, like so many other characters in Tai Tham is that they style of U+1A7C KHUEN-LUE KARAN is very similar to the style of U+1A58 MAI KANG LAI in Lanna. Thus if the wrong style were used, there would be confusion over which character was being referred to. To allow for more unified fonts, it was decided to keep U+1A7C KUEN-LUE KARAN only for use in Kheun and U+1A7A RA HAAM only for use in Lanna. Kheun does make use of U+1A7A RA HAAM for an archaic final ra. U+1A58 MAI KANG LAI is styled differently for Lanna versus Kheun.

2.12 Cryptogrammic Dot

The U+1A7F (◌ CRYPTOGRAMMIC DOT) is used to modify consonants in a standard cryptographic pattern where the number of dots says how many characters to advance the base character through the sequence to find the actual letter this character represents. There are often multiple dots, and they can occur with other diacritics:

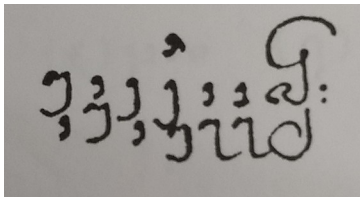


This calls for the cryptogrammic dot to be a consonant modifier.

CC = C CM* M? V? T? F?
CM = \u1A7F

2.13 Digits

Digits can act as consonants and are used in contractions or in cryptogrammic text:



As such it makes sense to add both sets of digits to the consonants class:

C = [\u1A20-\u1A54\u1A80-\u1A89\u1A90\u1A99] [\u1A5A\u1A5B]? | \u1A49 \u1A60 [\u1A20-\u1A3E]

2.14 Extra Consonants

There are various extra consonants in the range U+1A57, U+1A59 – U+1A5E that need to be accounted for. In addition U+1A6D still requires a home.

The characters U+1A57 (◌᩵ SIGN LA TANG LAI) (strictly a ligature of subjoined la and subjoined nga), U+1A5C (◌᩼ SIGN MA), U+1A5D (◌᩽ SIGN BA) and U+1A5E (◌᩾ SIGN SA) are merely alternative forms of subjoined characters to be used when they contrast with the normal subjoined forms, and therefore can be used wherever such subjoined consonants go. Although, in these cases they do not occur following a U+1A49 (ᩉ HA), so are limited to finals. U+1A59 (◌᩹ SIGN FINAL NGA) is also a final. U+1A6D (◌᩵ SIGN OY) is a combination of vowel and final so we add it as a final.

F_s = [\u1A4B\u1A54\u1A6D] | \u1A60 [\u1A20-\u1A4A]
F = F_s? [\u1A57-\u1A59\u1A5C-\u1A5E\u1A74\u1A7B]? (\u1A60 S)?

U+1A5B (◌᩻ HIGH RATHA OR LOW PA) is used in conjunction with another consonant even when the consonant is subjoined. Thus we can add it to the consonant specification. Likewise, U+1A5A (◌᩺ SIGN LOW PA) only occurs in one word in Tai Lue. And while it is stored after a base consonant, it is sounded before it. Therefore we add it to the initial.

C = ([\u1A20-\u1A54\u1A80-\u1A89\u1A90-\u1A99] [\u1A5A\u1A7B]? | \u1A49 \u1A60 [\u1A20-\u1A3E]) \u1A5B?

2.15 Summary

In summary our syllable description is:

S = CC (V_s | V_f)? SM?
CC = C CM* M? V? T? F?
C = ([\u1A20-\u1A54\u1A80-\u1A89\u1A90-\u1A99] [\u1A5A\u1A7B]? | \u1A49 \u1A60 [\u1A20-\u1A3E]) \u1A5B?
CM = \u1A7F
M = \u1A55? \u1A56? (\u1A60\u1A45)? (\u1A60\u1A3F)?
V = V_p? V_b? V_a?
V_p = [\u1A6E-\u1A72]
V_b = [\u1A69-\u1A6A\u1A6C]
V_a = [\u1A65-\u1A68\u1A6B\u1A73] [\u1A62\u1A74]?
T = [\u1A75-\u1A79]
F_s = [\u1A4B\u1A54\u1A6D] | \u1A60 [\u1A20-\u1A4A]
V_s = AV F_s?
AV = [\u1A63\u1A64] \u1A74?

V_f = \u1A4B? \u1A61
F = F_s? [\u1A57-\u1A59\u1A5C-\u1A5E\u1A74\u1A7B]? (\u1A60 S)?
SM = [\u1A7A\u1A7C]? \u1A7F?

Notice that we have not tried to ensure this regular expression works with normalized text since this is purely for descriptive purposes.

3 Visually Motivated Ordering

The problem with the above scheme, even fully filled out to describe every situation, is that it requires someone to be able to read a word before they can type it. The prevalence of mistyping is huge and the mistypings are usually not visible to the typist and so cannot be caught, except by spell checking. And then all the poor user sees a red squiggly line or whatever indicates a spelling error, while the word looks perfectly OK to them, and is visually correct. Consider this little gem:

ᩉᩮᨶᨲ	ᩉᩤᨶᩮᨲ	1A49 (ᩉ HA) 1A60 SAKOT 1A36 (ᩃ NA) 1A65 (ᩤ I) 1A3E (ᩮ MA) | 1A20 (ᨠ KA) 1A41 (ᨲ RA) 1A49 (ᩉ HA) 1A65 (ᩤ I) 1A60 SAKOT 1A36 (ᩃ NA) | 1A3E (ᩮ MA) 1A20 (ᨠ KA) 1A41 (ᨲ RA)

With | separating the linguistic syllables. Both readings are correct. Also, by analagous environments:

Id	Lanna	Khuen	Thai	Encoding	Encoding	Thai	Khuen	Lanna
1.	ᩉᩣ	ᩉᩤ	หนี	1A49 (ᩉ HA) 1A60 SAKOT 1A36 (ᩃ NA) 1A66 (ᩦ II)	1A49 (ᩉ HA) 1A65 (ᩤ I) 1A60 SAKOT 1A36 (ᩃ NA)	หิน	ᩉᩤ	ᩉᩣ
2.	ᨠᩯ	ᨠᩮ	เกว	1A20 (ᨠ KA) 1A6E (ᩮ E) 1A60 SAKOT 1A45 (ᩯ WA)	1A20 (ᨠ KA) 1A60 SAKOT 1A45 (ᩯ WA) 1A6E (ᨦᩮ E) 1A41 (ᨲ RA)	เกวน	ᨠᩮᨶ	ᨠᩯᨶ

But it must be born in mind that the text itself is ambiguous. By asking a user to type the correct reading of the text, we are asking them to enter more information than is in the text itself. Tai Tham is not unique in this respect, but it is recognised as a problem in all areas where it occurs. Why should we expect a typist to disambiguate the ambiguous spelling? Can we find an encoding scheme that truly represents the text as it is presented rather than trying to encode some underlying interpretation of the text? This is a key principle:

> *Unicode encodes text and not underlying morphophonemics*

This does not preclude encoding text in such a way as to aid its processing, but where there is ambiguity in the text, we should expect to see ambiguity in the encoding. Can we therefore come up with an encoding scheme that is “mostly right” and acknowledge that there will be situations where the encoding falls on the wrong side of an ambiguity?

One option is to standardise on some of the ambiguities regarding subjoined consonants as to whether they are syllable chaining, medials or finals. If we can get the order right and there is no impact on rendering, then we would gain a consistent storage order which would make implementing keyboards and many other things much easier / doable. On the other hand we would probably mess up collation. But since all the collation schemes presented for Tai Tham based languages involve being able to read the word in order to collate it, this is no bad thing to consider collation from scratch! But it does need to be born in mind that a switch to a visual encoding will make algorithmic analysis much harder for Tai Tham script. Getting from a visual encoding to a linguistically accurate one is not easy and will probably require dictionary approaches as well as algorithmic.

3.1 Encoding Order

So far, our analysis has been in terms of linguistic syllables. But a visually based interpretation works in terms of grapheme clusters. A grapheme cluster consists of a base character surrounded by non-spacing diacritics. This is a more general definition than is strictly defined for Tai Tham in UAX 29. In Tai Tham, we stretch even this a little in many ways, but the basic principle applies. A Tai Tham linguistic syllable may consist of multiple grapheme clusters and a grapheme cluster may represent more than one linguistic syllable.

The easiest encoding order to work with is where any diacritic in a grapheme cluster has a fixed position in that order. We will do our best to achieve that, and treat as special cases those situations where Tai Tham cannot fit that model. There are three grapheme cluster types in Tai Tham:

1. A consonant based cluster, based around a consonant and with diacritics attaching to that base or each other.
2. A spacing vowel cluster with a base of U+1A63 (ᩣ AA) or U+1A64 (ᩤ TALL AA) with diacritics attaching to it or each other.
3. A simple spacing vowel U+1A61 (ᩡ A), which takes no diacritics.

Given that the encoding requirement is that two different sequences will look different, we are not concerned with ensuring that crazy sequences, such as ensuring that illegal diacritics added to U+1A61 ᩡ A, are marked as erroneous. It is then possible to merge all these cluster types into a single cluster. This resolves many of the questions of what can and cannot follow a spacing vowel.

The precise order that diacritics are stored following a base can be arbitrary so long as everyone agrees on that order and that shapers and fonts know how to get from that order to the intended visual rendering. Thus, for example, subjoined consonants may be stored early in the sequence or late. We choose early because that aligns itself more easily with other scripts and the needs of the USE (Universal Shaping Engine). Linguistically, this is not ideal, but a visual encoding is not intended to be linguistically accurate.

Based on these principles we can create an initial regular expression:

```
1   S = C CM? (SUB CM?)* M? Vp? Va* (Vb SUB*)? T* F?
2   C = [\u1A20-\u1A54\u1A61\u1A63\u1A64\u1A80-\u1A89\u1A90-\u1A99]
3   CM = \u1A5B? \u1A7F*
4   SUB = \u1A60 [\u1A20-\u1A4A\u1A53\u1A54] | [\u1A56\u1A57\u1A5C-\u1A5E]
5   M = [\u1A55\u1A5A]
6   Vp = [\u1A6E-\u1A72]
7   Va = [\u1A62\u1A65-\u1A68\u1A6B\u1A73\u1A74]
8   Vb = [\u1A69-\u1A6A\u1A6C\u1A6D]
9   T = [\u1A58\u1A59\u1A75-\u1A79\u1A7A\u1A7C]
10  F = \u1A7B
```

This definition takes some explaining.

Line 1 gives the primary cluster structure. V_b and V_a have been reordered to conform to USE order. Although in general usage, this is not preferred. In the USE multiple vowel diacritics are allowed and therefore we simplify the definition of V_a in line 7 and simply allow multiple. Since below vowels and subjoined consonants can interact visually, particularly in Khuen, it is necessary to allow subjoined consonants after below vowels.

Line 3 lists the consonant modifiers. U+1A5A (ᩚ PA) occurs only in one rare word and occurs only once. Multiple occurrences are expected to stack vertically and so be evident. U+1A5B (ᩛ RATHA OR PA) also only occurs once. U+1A7F (᩿ CRYPTOGRAMMIC DOT) may well occur multiple times under a character and stacks downward.

Line 4 is the list of subjoined consonants which may interact visually, vertically (or in the case of Khuen and some Lanna, horizontally). Whether the interaction is visible is based on font preference. But since some fonts to require appropriate order for visual interaction, the order needs to be the order they would be if they did interact visually.

Line 5 contains the sole medial. The reason that U+1A55 (ᩕ MEDIAL RA) is a medial and not a subjoined character is that a cluster can have only one.

Line 7. U+1A74 (᩠ MAI KANG) interacts with other above vowels. U+1A73 (᩵ OA ABOVE) is a vowel in contrast to U+1A77 (᩷ TONE-3) which is a tone mark which only occurs in Khuen, and may occur after an above vowel.

Line 8 describes below vowels. Due to a contrast between words like ᩉᩣ and ᩉᩤ, it is necessary to allow a SUB after a lower vowel as well as before it. But this contrast is only needed because of the visual interaction between lower vowels and subjoined consonants. In addition, only true subjoined characters may be used there and not medials.

Line 9 shows that tones and other finals do interact. For example, compare ᩉᩰᩥ U+1A49 (ᩉ HA) U+1A6C (ᩬ OA) U+1A59 (ᩰ FINAL NGA) U+1A76 (᩶ TONE-2) compared to ᩉᩣᩰᩥ U+1A49 (ᩉ HA) U+1A76 (᩶ TONE-2) U+1A60 SAKOT U+1A38 (ᩣ PA) U+1A59 (ᩰ FINAL NGA).

Line 10 shows the final modifier.

The recursion is removed. Instead we change the syllable structure to reflect a grapheme cluster. We do this by treating a spacing final base as the start of a new cluster and then explicitly describe the subjoined finals via SUB.

3.1.1 Examples

Based on this structure, we give some example words along with their current and proposed spellings. Renderings are given in Lanna and Khuen style.

Id	Lanna	Khun	Original encoding	Proposed encoding
1.	ᨣᩴ	ᨠᩮ	1A20 (ᨣ KA) 1A65 (ᩴ I) 1A60 SAKOT 1A36 (᩠ NA)	1A20 (ᨣ KA) 1A60 SAKOT 1A36 (᩠ NA) 1A65 (ᩴ I)
2.	ᨧ᩶ᨶ	ᨧᨶ᩶	1A27 (ᨧ CA) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA) 1A74 (ᩃ MAI KANG)	1A27 (ᨧ CA) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA) 1A74 (ᩃ MAI KANG)
3.	ᨦᩮᩣ	ᨦᩮᩣ	1A41 (ᨦ RA) 1A6E (ᩮ E) 1A6C (ᩬ OA) 1A65 (ᩴ I) 1A75 (᩶ TONE-1) 1A60 SAKOT 1A3F (᩵ YA)	1A41 (ᨦ RA) 1A6E (ᩮ E) 1A65 (ᩴ I) 1A6C (ᩬ OA) 1A60 SAKOT 1A3F (᩵ YA) 1A75 (᩶ TONE-1)
4.	ᨡᩯ	ᨡᩯ	1A23 (ᨣ KA) 1A55 (ᩯ MEDIAL RA) 1A60 SAKOT 1A45 (᩠ WA) 1A6B (ᩫ O)	1A23 (ᨣ KA) 1A60 SAKOT 1A45 (᩠ WA) 1A55 (ᩯ MEDIAL RA) 1A6B (ᩫ O)
5.	ᨲᩮᨿ	ᨲᩮᨿ	1A34 (ᨲ TA) 1A58 (ᩮ MAI KANG LAI) 1A60 SAKOT 1A43 (᩠ᩃ LA) 1A63 (ᨶ AA) 1A60 SAKOT 1A3F (᩵ YA)	1A34 (ᨲ TA) 1A60 SAKOT 1A43 (᩠ᩃ LA) 1A58 (ᩮ MAI KANG LAI) 1A63 (ᨶ AA) 1A60 SAKOT 1A3F (᩵ YA)
6.	ᨦᨲᩩ	ᨦᨲᩩ	1A20 (ᨣ KA) 1A6E (ᩮ E) 1A60 SAKOT 1A32 (᩠ TA) 1A69 (ᩩ U)	1A20 (ᨣ KA) 1A60 SAKOT 1A32 (᩠ TA) 1A6E (ᩮ E) 1A69 (ᩩ U)
7.	ᨠᨶᩁ	ᨠᨶᩁ	1A20 (ᨣ KA) 1A6E (ᩮ E) 1a63 (ᨶ AA) 1a61 (ᩁ A)	1A20 (ᨣ KA) 1A6E (ᩮ E) 1a63 (ᨶ AA) 1a61 (ᩁ A)
8.	ᨦᩮᩣᨶ	ᨦᩮᩣᨶ	1A3E (ᨾ MA) 1A6F (ᩮᩮ EE) 1A76 (᩶ TONE-2) 1A60 SAKOT 1A36 (᩠ NA) 1A60 SAKOT 1A45 (᩠ WA) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA)	1A3E (ᨾ MA) 1A60 SAKOT 1A36 (᩠ NA) 1A60 SAKOT 1A45 (᩠ WA) 1A6F (ᩮᩮ EE) 1A76 (᩶ TONE-2) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA)
9.	ᨧᩣᨶ	ᨧᩣᨶ	1A3B (ᨧ PA) 1A6D (᩵ OY) 1A60 SAKOT 1A45 (᩠ WA) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA)	1A3B (ᨧ PA) 1A6D (᩵ OY) 1A60 SAKOT 1A45 (᩠ WA) 1A75 (᩶ TONE-1) 1A63 (ᨶ AA)
10.	ᨣᩱ	ᨣᩱ	1A35 (ᨤ THA) 1A64 (ᩮ TALL AA) 1A60 SAKOT 1A32 (᩠ TA) 1A69 (ᩩ U)	1A35 (ᨤ THA) 1A64 (ᩮ TALL AA) 1A60 SAKOT 1A32 (᩠ TA) 1A69 (ᩩ U)

1. This is a very common syllable structure CVC and is probably the most obvious difference between a visual and logical encoding. The subjoined na is encoded visually in an arbitrary (by linguistic standards) but consistent order.
2. This word might be expected to differ between encodings but doesn’t.
3. Only the tone moves here. This is because of the lower vowel that allows the final to be stored after it. This is because there can be a visual difference between a lower vowel before or after a final. But notice that the tone is always at the end.

4. What is noticeable here is that medials are now encoded after subjoined characters.
5. U+1A58 here is stored and renders before the U+1A63 (ᩣ AA) vowel.
6. The linguistics of the proposed encoding here, are a bit of a mess, but the rendering is fine.
7. Here U+1A61 (ᩡ A) is a simple base character. And there are words like ᨷᩕᩡ᩠ᨿᩣ U+1A37 (ᨷ BA) U+1A55 (◌ᩕ MEDIAL RA) U+1A61 (ᩡ A) U+1A60 SAKOT U+1A3F (◌᩠ᨿ YA) U+1A63 (ᩣ A) (Lao found in [1]) that also work.
8. An example of a polysyllabic cluster with two tone marks.
9. Another polysyllabic cluster: boywa. Due to the awkwardness of the combining orders, the original encoding is particularly nasty here.
10. There is no difference in encoding here, but the sequence is interesting.

3.2 Under Differentiation

In this section we consider the proposed encoding for issues with under differentiation.

Whereas the original encoding suffered from over differentiation in that two visually identically strings could be spelled differently, this new encoding has problems with under differentiation where necessary visual contrasts cannot be represented in the encoding. The following table shows examples of where there may be problems.

	lanna	khün	thai	original encoding	visual encoding
1.	ᨡ᩠᩻ᨿᩡ	ᨡ᩠ᨿ᩻ᩡ	ชยะ	1A21 (ᨡ KHA) 1A7B (◌᩻ MAI SAM) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A61 (ᩡ A)	1A21 (ᨡ KHA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A7B (◌᩻ MAI SAM) 1A61 (ᩡ A)
2.	ᩉ᩠ᨿᩬᩴ᩶	ᩉ᩠ᨿᩳ᩶	หย้อ	1A49 (ᩉ HA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A6C (◌ᩬ OA) 1A74 (◌ᩴ MAI KANG) 1A76 (◌᩶ TONE-2)	1A49 (ᩉ HA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A74 (◌ᩴ MAI KANG) 1A6C (◌ᩬ OA) 1A76 (◌᩶ TONE-2)
3.	ᩉᩬ᩠ᨿ	ᩉᩬ᩠ᨿ	หอย	1A49 (ᩉ HA) 1A6C (◌ᩬ OA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)	1A49 (ᩉ HA) 1A6C (◌ᩬ OA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)
4.	ᨨ᩠᩻ᩃᩬᨦ	ᨨ᩠ᩃᩬ᩻ᨦ	ฉลอง	1A28 (ᨨ CHA) 1A7B (◌᩻ MAI SAM) 1A60 SAKOT 1A43 (◌᩠ᩃ LA) 1A6C (◌ᩬ OA) 1A26 (ᨦ NGA)	1A28 (ᨨ CHA) 1A60 SAKOT 1A43 (◌᩠ᩃ LA) 1A6C (◌ᩬ OA) 1A7B (◌᩻ MAI SAM) 1A26 (ᨦ NGA)
5.	ᨣᩩ᩠ᨿ	ᨣᩩ᩠ᨿ	คุย	1A23 (ᨣ KHA) 1A69 (◌ᩩ U) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)	1A23 (ᨣ KHA) 1A69 (◌ᩩ U) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)
6.	ᩋᩣᩴᨶᩣ᩠ᨲ	ᩋᩣᩴᨶ᩠ᨲᩣ	อำนาต	1A4B (ᩋ A) 1A63 (ᩣ AA) 1A74 (◌ᩴ MAI KANG) 1A36 (ᨶ NA) 1A63 (ᩣ AA) 1A60 SAKOT 1A32 (◌᩠ᨲ TA)	1A4B (ᩋ A) 1A63 (ᩣ AA) 1A74 (◌ᩴ MAI KANG) 1A36 (ᨶ NA) 1A60 SAKOT 1A32 (◌᩠ᨲ TA) 1A63 (ᩣ AA)

1. The U+1A7B (◌᩻ MAI SAM) here indicates a syllable break and that the following U+1A60 SAKOT U+1A3F (◌᩠ᨿ YA) is the start of a new syllable `chaya`. The visual encoding treats it as if the U+1A60 SAKOT U+1A3F (◌᩠ᨿ YA) were double acting or that the cluster was the last in a reduplicated word. The question is whether there needs to be a visual rendering contrast between the two possible encoding positions of the MAI SAM. Double acting consonants are rare, which would mean that the mai sam in this position is indicating a syllable break. But this mean is really only indicated because of the following U+1A61 (ᩡ A), which if missing would imply the MAI SAM is a reduplicating mark.
2. There is very little difference in encoding between the original and visual encoding, here. The difference here calls for some diacritic reordering during rendering. This is the only example in which the Khuen spelling and encoding is different. It is spelled U+1A49 (ᩉ HA) U+1A60 SAKOT U+1A3F (◌᩠ᨿ YA) U+1A73 (◌ᩳ OA ABOVE) U+1A76 (◌᩶ TONE-2) with no difference between the original or visual encoding.

3. There is no difference between the visual and the original encoding. In Khuen this word would be spelled ꪽꪵ U+1A49 (ꪽ HA) U+1A6D (ꪵ OY). In general in Khuen, if there is a sequence of below diacritics and one of them has a tall stem, that is moved to the front.
4. This is a near identical problem to example 1.
5. This example is awkward. The phonetic and vertically stacked spelling calls for the U+1A69 (ꪹ U) to come before the subjoined ya, but in Khuen the u is clearly visually located after the subjoined ya. Further, in Lanna, there are examples of the u being below the subjoined ya. The contrast is needed. Compare ꪽꪻ U+1A49 (ꪽ HA) U+1A69 (ꪹ U) U+1A60 SAKOT U+1A3F (ꪻ YA) vs ꪽꪻꪀ U+1A49 (ꪽ HA) U+1A60 SAKOT U+1A3F (ꪻ YA) U+1A69 (ꪹ U) U+1A76 (ꪶ TONE-2) U+1A26 (ꪀ NGA), which do not contrast visually in Khuen with ꪽꪻ and ꪽꪻꪶ.
6. The difficulty here is that the U+1A63 (ꪣ aa) has no space and so we have no idea what happens underneath it. Perhaps the easiest solution is to always use U+1A36 (ꪖ NA) U+1A63 (ꪣ AA) immediately adjacent to each other as if there was a ligature spelling for that character.

As seen in the above examples, U+1A7B (ꪾ MAI SAM) is problematic. There can be needs for contrasting rendering of the diacritic. In the case where it marks a syllable break or a double acting character, it may well be moved further left, more over the base. The final use as a reduplication mark can call for it to be pushed as far right as possible. The question is, do we need to represent this distinction and if so, how?

The need to contrast spellings where a subjoined consonant may render before or after a lower vowel (as in examples 2 and 3 above) is liable to be somewhat problematic. Because there is a contrast of order allowed in the encoding, it is not possible for the keyboard to resolve that order automatically, the way it can for most diacritics. Lack of visual differentiation is often lacking in fonts and may require users to have to think more deeply to type in the correct order, since a required difference in order results in no visual distinction.

4 Refined Visual Encoding

The proposed encoding in the previous section has some significant weaknesses:

- The encoding contrast between a subjoined character and a below vowel forces users to make a distinction that may not be visible to them or that they do not know how to resolve.
- The encoding is not refined enough to allow fonts that wish to contrast the position of tone marks or upper vowels in relation to spacing subjoined characters, like U+1A60 SAKOT U+1A3F (◌᩟ YA). Which is supported by the current encoding scheme. For example compare ᨦ᩠ᨿ

The key aspect of these issues is that neither of them actually affect spelling. The contrasts are purely visual and while there are minimal pairs by analagous environment, in the case of the below vowel, many fonts under differentiate to the point where it is sufficient to under differentiate in plain text. Therefore it is not required for a typist to provide differentiation or for computing processes to rely on contrast in these ways. By purely using order to mark visual differentiation, we are requiring users to type the contrast, which they may not see. An alternative approach to encoding contrast through ordering is to encoding contrast through extra characters. These characters are not there to provide spelling contrast but to enable layout contrasts to be encoded if so desired. As such they are format characters.

We start with a simplified version of the encoding order presented in the previous section:

```
1   S = C CM? SUB* M? Vp? Va* Vb? T* F?
2   C = [\u1A20-\u1A54\u1A61\u1A63\u1A64\u1A80-\u1A89\u1A90-\u1A99]
3   CM = \u1A5B? \u1A7F*
4   SUB = \u1A60 [\u1A20-\u1A4A\u1A53\u1A54] | [\u1A56\u1A57\u1A5C-\u1A5E]
5   M = [\u1A5A\u1A55]
6   Vp = [\u1A6E-\u1A72]
7   Va = [\u1A62\u1A65-\u1A68\u1A6B\u1A73\u1A74]
8   Vb = [\u1A69-\u1A6A\u1A6C\u1A6D]
9   T = [\u1A58\u1A59\u1A75-\u1A79\u1A7A\u1A7C]
10  F = \u1A7B
```

The difference here is that we have removed the V_b SUB sequence. How are we, therefore, to represent the necessary contrast? We introduce two new format characters:

U+1A8E	TAI THAM SIGN INITIAL
U+1A8F	TAI THAM SIGN FINAL

Both of these characters are of category Cf and are stored as CM in the encoding order. As such they are formatting controls placed following a subjoined character. We start with a discussion of U+1A8F SIGN FINAL. This may follow a subjoined character and indicates that this and all following subjoined characters are finals in the cluster. Thus any following vowels are to be rendered ‘before’ the marked subjoined character. It also provides indication for those fonts that differentiate the position of vowels or tones in relation to spacing subjoined characters. This table shows two similar words. The first column shows vowel positional contrasts, while the second shows them ignored. The original encoding is given and then a proposed explicit encoding. If the two format characters are removed from the proposed strings, one would expect to see something equivalent to the simple rendering.

Contrast	Simple	Original	Proposed explicit
ᩉ᩠ᩈ	ᩉ᩠ᩈ	1A49 (ᩉ HA) 1A60 SAKOT 1A3F (◌᩟ YA) 1A6C (◌ᩬ OA) 1A74 (◌ᩴ MAI KANG) 1A76 (◌᩶ TONE-2)	1A49 (ᩉ HA) 1A60 SAKOT 1A3F (◌᩟ YA) 1A8E SIGN INITIAL 1A6C (◌ᩬ OA) 1A74 (◌ᩴ MAI KANG) 1A76 (◌᩶ TONE-2)

ᩉ᩠ᨿᩬ᩶	ᩉᩬ᩠᩶ᨿ	1A49 (ᩉ HA) 1A6C (◌ᩬ OA) 1A76 (◌᩶ TONE-2) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)	1A49 (ᩉ HA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A8F SIGN FINAL 1A6C (◌ᩬ OA) 1A76 (◌᩶ TONE-2)

Where there is no explicit use of format controls, the rendering of the text is up to the heuristics of the font.

The characters are format controls and are optional. For purposes of spell checking, they are ignored. Having said that, they are strongly encouraged due to the value in appropriate rendering contrasts for those fonts that show such contrasts. The rules for the automatic insertion of such marks are beyond the capabilities of a font or a shaping engine, requiring, in some cases, a dictionary. Thus they may be inserted using spell checking techniques.

The second character to consider is U+1A8E SIGN INITIAL. If one has a consonant followed by a spacing subjoined character, one does not know whether the subjoined character is definitely not a final or simply has not been marked as a final and should have been. U+1A8E SIGN INITIAL allows the former case of this character not being a final, to be clearly marked. The use of two such marks overcomes the problem that is also prevalent in Indic text that a system does not know whether the absense of a ZW(N)J indicates the desire for a conjunct or whether a more sophisticated font with more conjuncts would render the text differently from that desired.

Another use for these formatting controls is the positioning of U+1A7B (◌᩻ MAI SAM). As described, there are 3 functions for the character. Some fonts show a positional contrast between the use of mai sam as a reduplication mark and for its other uses. For example:

Contrast	**Original**	**Proposed explicit**
ᩈ᩠ᨿᩬ᩻	1A48 (ᩈ SA) 1A6C (◌ᩬ OA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A7B (◌᩻ MAI SAM)	1A48 (ᩈ SA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A8F SIGN FINAL 1A6C (◌ᩬ OA) 1A7B (◌᩻ MAI SAM) 1A8F SIGN FINAL
ᩈ᩠᩻ᨿᩬᨦ	1A48 (ᩈ SA) 1A7B (◌᩻ MAI SAM) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A6C (◌ᩬ OA) 1A26 (ᨦ NGA)	1A48 (ᩈ SA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A8E SIGN INITIAL 1A6C (◌ᩬ OA) 1A7B (◌᩻ MAI SAM) 1A8E SIGN INITIAL 1A26 (ᨦ NGA)
ᩮᨧ᩠᩻ᩃᩥ᩠ᨿ	1A27 (ᨧ HIGH CA) 1A7B (◌᩻ MAI SAM) 1A60 SAKOT 1A43 (◌᩠ᩃ LA) 1A6E (ᩮ◌ E) 1A65 (◌ᩥ I) 1A60 SAKOT 1A3F (◌᩠ᨿ YA)	1A27 (ᨧ HIGH CA) 1A60 SAKOT 1A43 (◌᩠ᩃ LA) 1A60 SAKOT 1A3F (◌᩠ᨿ YA) 1A8F SIGN FINAL 1A6E (ᩮ◌ E) 1A65 (◌ᩥ I) 1A7B (◌᩻ MAI SAM) 1A8E SIGN INITIAL

In the last row, one might expect to see U+1A8E SIGN INITIAL following the U+1A60 SAKOT U+1A43 (◌᩠ᩃ LA), but this is redundant due to the occurrence of U+1A8F sign final following the next subjoined consonant.

If these characters occur in a sequence context where they are undefined, they are ignored.

There are various reasons for not wanting to use ZW(N)J for these functions:

- ZW(N)J have grown their own specific meanings and behaviours not covered by these characters. For example ZWNJ causes a cluster break in the Universal Shaping Engine.
- It is not clear which function would be covered by which of ZW(N)J. Although an arbitrary choice would be acceptable.
- ZW(N)J can only occur following a halant/sakot in the USE. This would be useable for the subjoined marking but not for the mai sam.
- ZW(N)J may be used for ligation control within Tai Tham, for example: ᨬ᩠ᨬ U+1A2C (ᨬ NYA) U+1A60 SAKOT U+1A2C (◌᩠ᨬ NYA) vs ᨬ᩠‍ᨬ U+1A2C (ᨬ NYA) U+1A60 SAKOT U+200D ZWJ U+1A2C (◌᩠ᨬ NYA).

The reasons in favour of using ZW(N)J are:

- No need to introduce extra Cf characters.

- These characters may be useful for resolving similar issues with regard to Old Khmer, although the analysis of that writing system and its encoding has yet to be done. As such they would cross script and using common characters would ease that crossing.

By including two such marks, it leaves the interpretation of unmarked text up to the rendering system. It can be as sophisticated as it likes and where that sophistication breaks down, the format characters can be used to resolve the problem. Or a font may be unsophisticated and require a significant level of text marking, where it wants to show a visual contrast, if at all.

The advantage this system brings is that naive text entry is regularised for purposes of text comparison with the support of whatever level of sophistication a font may bring, while also catering to the needs of those requiring precise encoding for the presentation of high quality text.

Thus the encoding order becomes, with additions in italics:

1 S = C CM? SUB* M? V_p? V_a* V_b? T* F?
2 C = [\u1A20-\u1A54\u1A61\u1A63\u1A64\u1A80-\u1A89\u1A90-\u1A99]
3 CM = \u1A5B? \u1A7F*
4 SUB = \u1A60 [\u1A20-\u1A4A\u1A53\u1A54] *[\u1A8E\u1A8F]?* | [\u1A56\u1A57\u1A5C-\u1A5E]
5 M = [\u1A5A\u1A55]
6 V_p = [\u1A6E-\u1A72]
7 V_a = [\u1A62\u1A65-\u1A68\u1A6B\u1A73\u1A74]
8 V_b = [\u1A69-\u1A6A\u1A6C\u1A6D]
9 T = [\u1A58\u1A59\u1A75-\u1A79\u1A7A\u1A7C]
10 F = \u1A7B *[\u1A8E\u1A8F]?*

4.1 Issues

There are outstanding issues that may or may not need to be addressed:

- Contractions where one might see something like: ᯖᯩ U+1A48 (ᯎ SA) U+1A69 (◌ᯩ U) U+1A60 SAKOT U+1A43 (◌ᯖ LA) U+1A66 (◌᯦ II). Awaiting evidence of such text, although it may be argued that it is beyond scope and into the realms of calligraphy. The issue can be addressed either by allow U+1A8F SIGN FINAL following the U+1A66 vowel or by the introduction of yet another format control for the same purpose.

5 Implementation

Having proposed a visual encoding, we now examine how it might be implemented. The great value of a visual encoding is consistency, with on visual form having one encoding, which makes implementation easier. Although it can push other problems onto the font. Without it, identifying inconsistent sequences is nearly impossible. We first examine the issue of rendering and shaping and then look at the process of normalizing the order of diacritics follow base characters, both as part of text normalization but also as part of keyboarding, where we allow a relatively free typing order, but can ensure consistent output.

5.1 Shaping

A primary text rendering implementation is that of OpenType. In overly broad outline, this involves the structuring of text into clusters, based on regular expressions, much as we have been doing, and then applying lookups over the glyphs in a cluster. The primary structuring system in OpenType for Tai Tham is the Universal Shaping Engine. Can we fit the requirements for Tai Tham into the USE with minimal changes?

One advantage that rendering has over data entry is that it can be more relaxed about things occuring in the wrong order. So long as there is a visual difference between two different sequences, there is no need for the shaping engine to intervene and insert some kind of visual distinction (usually by adding a dotted circle somewhere in the sequence). Thus there is no need to stop particularly egregious sequences like U+1A61 A U+1A61 AA or the like.

Mapping from the visual encoding onto the USE is first a matter of mapping visual encoding structural classes onto USE classes:

C → B
CM(\u1A8E\u1A8F) → **CF**
CM (\u1A5B\u1A7F) → CMBlw
\u1A60 → H
SUB (\u1A56\u1A57\u1A5C-\u1A5E) → SUB
M (\u1A55)→ MPre
M (\u1A5A) → MAbv
V_p → VPre
V_a → VAbv
V_b → VBlw
T → VMAbv
F (\u1A58\u1A59\u1A7A\u1A7C) → FAbv
F (\u1A7B) → FM

5.1.1 USE Structure

The proposed structure extends the current USE syllable structure as marked in bold:

[< R | CS >] < B [VS] | GB > (CMAbv)* (CMBlw)* (< H B [VS] | SUB > (CMAbv)* (CMBlw)* **[CF]**)*
[MPre] [MAbv] [MBlw] [Mpst]
(VPre)* (VAbv)* (VBlw)* (VPst)*
(VMPre)* (VMAbv)* (VMBlw)* (VMPst)*
(FAbv)* (FBlw)* (FPst)* [FM **[CF]**]

5.1.2 OpenType Implementation

After the grapheme clusters have been identified there are still issues with ordering diacritics such that diacritic attachment tree is stored in the right order.

- U+1A74 MAI KANG may be stored following a spacing vowel, but in some cases need to be reordered onto the previous consonant, before even a tone on that consonant. But this is not true for all fonts.

- U+1A7F CRYPTOGRAMMIC DOT is stored late in the sequence and needs to be shifted to immediately following the initial consonant glyph.
- Subjoined consonants followed by U+1A8F SIGN FINAL need to be moved after the vowels. Alternatively what a vowel attaches to may be dependent on the presence of U+1A8F SIGN FINAL.
- Final superscript consonants (U+1A59, U+1A5A) will occur after the tone marks they should render below.

5.2 Normalization

How can we enforce a text to conform to the proposed encoding order described above? Unlike for rendering, the requirement here is to reorder a sequence of characters from an arbitrary keying order into a canonical spelling order, as described previously.

5.2.1 Canonical Ordering

There is a constraint on our preferred storage order and that is that marks with a Canonical Combining Class sort relative to their value after a character with CCC of 0. We must ensure that text that is ordered according to our order is not modified if normalized according to the Unicode Normalization algorithm. Thankfully, for Tai Tham, there is no difference between NFC and NFD.

It is most unfortunate that Tai Tham was landed with various diacritics being given somewhat arbitrary (in ordering terms) canonical combining classes. The overloading of classes as ordering means we have some real difficulties in the ordering of Tai Tham. And since it is impossible to fix these where they should be fix: in the standard. We need to go through some mental gymnastics to handle the resulting broken state of Tai Tham. The non-zero CCC values are:

\u1A60	9
\u1A7F	220
[\u1A75-\u1A79]	230
[\u1A7A-\u1A7C]	230

Non-zero CCC diacritics immediately preceding U+1A60 SAKOT are a problem because they are reordered immediately after the Sakot, between the Sakot and its following consonant. In the current scheme the only problem character is U+1A7F CRYPTOGRAMMIC DOT. By categorising this as CM it may occur immediately before a Sakot. But a cryptogrammic dot only modifies a base consonant. Therefore, on that understanding, it could occur anywhere in the cluster, even after a subjoined consonant, on the understanding that it is reordered during shaping to immediately after the consonant. Another constraint is that it has a CCC value less than that of VMAbv. Given we also want multiple of them, this limits us to a V of some kind or VMPre. We somewhat arbitrarily choose VMPre, since this is not a vowel. But then, neither is it a vowel modifier.

5.2.2 Sorted Ordering

Part of UTS 35, the CLDR file format description for locale information, describes keyboard layouts, and as part of that there is an algorithm given for reordering marks from their keyed order into the required underlying order. This algorithm may be simply extended to support text canonicalisation. There is no need to handle pre-vowels in such an algorithm.

One approach is to allocate every character in a string a given ordering number. Groups are those between a character with an order of 0 and the next character with order 0, while only including the first character with an order of 0. The characters are then ordered first by their order number and second by their current order (or index) in the string. The order numbers are not necessarily given by a simple lookup, but characters may given an order number based on a contextual match. All characters in the match are given the same order number (and so will keep their relative order). Character sequences are matched using a regular expression, which may include look ahead or look behind assertions. Unfortunately for most regular expression engines,

the look behind assertions are not necessarily fixed length (and so in implementation may need to be extracted and processed as another constraint on the match). All unmatched characters are given an order of 0. Negative order values indicate secondary sort relationships.

Character classes		Order
[\u1A20-\u1A54\u1A61\u1A63\u1A64]	→	0
[\u1A8E\u1A8F]	→	-2
[\u1A5B\u1A7F]	→	-3
\u1A60[\u1A20-\u1A4A\u1A4C]	→	10
\u1A60	→	10
[\u1A56\u1A57\u1A5C-\u1A5E]	→	15
\u1A5A	→	17
\u1A55	→	20
[\u1A6E-\u1A72]	→	60
[\u1A62\u1A65-\u1A68\u1A6B\u1A73\u1A74]	→	65
[\u1A69\u1A6A\u1A6C\u1A6D]	→	70
[\u1A75-\u1A79]	→	85
[\u1A58\u1A59\u1A7A\u1A7C]	→	105
\u1A7B	→	115

The values of the numbers are unimportant just their relative value, except the value of 0 which indicates the start of a new sequence order (and is the default for all unspecified codepoints, making the first list strictly redundant). Notice that at this point all reference to any guiding regular expressions are lost. The categories in such regular expressions are only helpful in arriving at a relative order.

The biggest problem with this implementation is that any tone [\u1A75-\u1A79] occurring after a spacing vowel is not reordered before it, as it should. This is a common problem with people often typing ‘am’ followed by a tone mark (which needs to be reordered across the ‘aa’ vowel). To support this one needs greater complexity. We, in effect, treat the spacing vowels [\u1A61\u1A63\u1A64] as diacritics onto a consonant based grapheme cluster, for the purposes of ordering.

Character classes		Order
[\u1A20-\u1A54]	→	0
[\u1A5B\u1A7F]	→	-2
[\u1A8E\u1A8F]	→	-3
\u1A60[\u1A20-\u1A4A\u1A4C]	→	10
\u1A60	→	10
\u1A60[\u1A75-\u1A79][\u1A20-\u1A4A\u1A4C]	→	10, 85, 10[3]
[\u1A56\u1A57\u1A5C-\u1A5E]	→	15
\u1A5A	→	17
\u1A55	→	20
[\u1A6E-\u1A72]	→	60
[\u1A62\u1A65-\u1A68\u1A6B\u1A73\u1A74]	→	65
[\u1A69\u1A6A\u1A6C\u1A6D]	→	70
[\u1A75-\u1A79]	→	85

3 This addressed normalized text and specifies an order for each character in the matched sequence.

Character classes		Order
[\u1A58\u1A7B](?=[\u1A55-\u1A57\u1A59\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7A\u1A7C-\u1A7F]*[\u1A61\u1A63\u1A64])	→	90
[\1A61\u1A63\u1A64]	→	100
(?<=[\u1A61\u1A63\u1A64][\u1A55-\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7F]*)\u1A60[\u1A20-\u1A49]	→	105
(?<=[\u1A61\u1A63\u1A64][\u1A55-\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7F]*)\u1A60	→	105
(?<=[\u1A61\u1A63\u1A64][\u1A55-\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7F]*)[\u1A56\u1A57\u1A5C-\u1A5E]	→	107
(?<=[\u1A61\u1A63\u1A64][\u1A55-\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7F]*)[\u1A65-\u1A68\u1A6B\u1A73\u1A74]	→	110
(?<=[\u1A61\u1A63\u1A64][\u1A55-\u1A5E\u1A62\u1A65-\u1A68\u1A6D-\u1A7F]*)[\u1A61-\u1A64\u1A69\u1A6A\u1A6C\u1A6D]	→	112
[\u1A58\u1A59\u1A7A\u1A7C]	→	115
\u1A7B	→	117

The grey cells indicate characters that may be followed by a tertiary sequence. The kline star * is limited for implementation purposes at 4 occurrences. The (?<= special regular expression syntax indicates a look behind constraint. In effect it says that the characters to receive the order must be preceded by the given subexpression.

In the order 110 vowel list, we remove \u1A62. U+1A62 (A) has a secondary linguistic function as a final k. But this character does not occur aftger U+1A63 (AA) and so it helps if a keyboard reorders them.

There is a possible ambiguity that if someone were to type a U+1A5B (RATHA OR PA) following a U+1A7B (MAI SAM), with no intervening base or subjoined consonant, the U+1A5B would be incorrectly ordered. But this is highly unlikely and users can be trained to resolve this corner case.

Missing from this list are CGJ, ZWJ, ZWNJ and variation selectors which all would take a tertiary order of -1.

5.3 Format Controls

A basic keyboarding order has already been described in the previous section. But with the move away from a logical encoding towards something much more regular and visual, more control load is placed on the keyboard. In particular the question of whether the keyboard can insert format controls automatically to reduce the interpretive load on the font which has much more limited processing capability.

It is of course possible to write a language specific keyboarding application that can even incorporate known standard spellings to help resolve format control insertion. But using more generic approaches can still yield helpful results. For example, it is possible, with care, for a keyboard to incorporate keying order to allow a user to specify whether, for example, a lower vowel should render before or after a subjoined consonant. It may not be 100% successful but can give a good default. In addition a good keyboard layout should provide the user with the capability to insert format controls if they so desire.

Where no format controls occur in a string, there are some basic principles that can be used to help provide some basic rendering. These rules should not be considered in any way normative. Consider S to be a subjoined substring: U+1A60 SAKOT followed by a consonant. Likewise MS is U+1A7B (MAI SAM). The subscript f indicates the marking of the text with a U+1A8F SIGN FINAL, and i with a U+1A8E SIGN INITIAL.

U+1A49 S → S_i
S S → S_i S_f
S S_i → S_i S_i
S_f ... MS → S_f ... MS_f
S_i ... MS → S_i ... MS_i

S [\u1A65-\u1A68] → S_f [\u1A65-\u1A68]
Fallback S → S_i

6 Acknowledgements

Thanks go to Richard Wordingham for his careful review of this document and some examples. He also suggested a much simpler grapheme cluster structure. Andrew Glass has worked hard to help integrate Tai Tham into the USE and opened my eyes to the idea that would could change the core sequence.

Thanks go to Payap University Linguistics Institute, Chiang Mai, Thailand, under whose auspices this work is done.

7 Bibliography

Due to the multiplicity of scripts involved, entries are sorted by date rather than author.

1. อรุณรัตน์ วิเชียรเขียว และคณะ (Arunradt Wichienkiaw et al) 1996 ᨻᨧᨶᩣᨶᩩᨠᩕᩫ᩠ᨾᩆᩢᨻ᩠ᨴ᩺ᩃᩣ᩠ᨶᨶᩣᨩᩮᨻᩣᩡᨣᩴᩤᨴᩦ᩵ᨷᩕᩣᨠ᩠ᨭᩱᨶᩱᨷᩃᩣ᩠ᨶ (พจนานุกรมศัพท์ลานนาเฉพาะคำที่ปรากฏในใบลาน The Northern Thai Dictionary of Palm-Leaf Manuscripts) ISBN 974-7047-77-2
2. สนั่น ธรรมธิ (Sanan Tamti) 2001. ยุตตสาระ หนึ่งในอักษล้านนา (Yutotsara Nuengnaiaksonlanna) ISBN 974-657-332-2
3. อุดม รุ่งเรืองศรี (Udom Rungruengsri) 2004. ᨻᨧᨶᩣᨶᩩᨠᩕᩫ᩠ᨾᩃ᩶ᩣ᩠ᨶᨶᩣ-ᨴᩱ ᨨᨷᩢ᩠ᨷᨾᩯ᩵ᨼ᩶ᩣᩉᩖ᩠ᩅᨦ (พจนานุกรมลานนา-ไทย ฉบับแม่ฟ้าหลวง Phojananukrom Lanna-Thai: Chababmefahluang) ISBN 974-386-044-4
4. เกษม ศิริรัตน์พิริยะ (Kasem Siritpiriya) 2005. ตั๋วเมือง การเรยนภาษาลานนาผ่านโครงสร้างคำ (Duamuang: Kanrianphasalannaphankhrongsangkham) ISBN 974-9942-00-0
5. Everson M. and Hosken M. 2006. *Proposal for encoding the Lanna script in the BMP of the UCS* (Unicode L2/06-258, ISO/IEC JTC1/SC2/WG2 N3121)
6. ပဋိဘာၼော ဘိက္ခု (Patibhano Bhikku) 2008. ဝစၼၚုကွံ်ဝဲမ္ယ-ဘ္ဂကြိ္သ (Tai-English Dictionary) ISBN 974-623-711-6
7. 2017. ᨻᩕᨠᩕᩥᩔᨴᨵᨾ᩠ᨾ᩼ᨣᨾ᩠ᨽᩦᩦ ᨽᩣᨣᨻᨶ᩠ᨵᩈᨬ᩠ᨬᩣᩉᩲ᩠ᨾ᩵ (พระคริสตธรรมคัมภีร์ภาคพันธสัญญาใหม่ Phrakhristthamkhamphi Phakphanotsanyamai)
8. ဇၢႆးၸေႈၸဵၼ် ၸေႃႇၵုၵ (Jaichechen Tengok) ဘဘိဓါၼ်သ္ပ်ၚ်ဝဟၼ္ဗေႂ် ဘၢသၢၶိုဝ် (Aphithansabwaihanble phaasaakhuen)
9. Unicode Technical Committee Unicode Technical Standard #35: Unicode Locale Data Markup Language (LDML) http://unicode.org/reports/tr35/tr35-keyboards.html#Element_reorder