An Introduction to Syllabaries and Syllabic Pattern Matching

Daniel Yacob, Ge'ez Frontier Foundation

A Simple Syllabary

Simple Syllabary						
	a	**e**	**i**	**o**	**u**	**+**
d						
l						
m						
n						
+						

In simplest terms a syllabary is a system of writing where each letter represents a syllable. Syllabaries under Unicode include Ethiopic, Cherokee, Canadian Unified Syllabics, Hiragana, Katakana and Yi. Most of these syllabaries are "Open Syllabaries" which means that the final consonant is absent in the consonant-vowel-consonant (CVC) sequence of a complete syllable. So, most syllables are in fact just a CV pattern and in some cases will even be just a lone "C" or "V" letter as in an alphabetic writing system. Syllabaries such as Devanagari or Hangul use a 2nd symbol, called a diacritic, to indicate the vowel. These syllabaries, Alpha Syllabaries, are not our concern here. Our focus in this article are those syllabaries that in Unicode do *not* use a separate character code to indicate the vowel component of a syllable.

An example syllabary that we will work with is the fictional "Simple Syllabary" shown in our first table on the left. Our simple syllabary is made up of basic geometric shapes and shaded in some way to indicate the presence of an implicit vowel. The table shows the phonetic mappings of the symbols to a few English consonants and the five basic vowels of English. Notice that the last row does not have a leading consonant. These "syllables" are just the simple case of the lone vowels. Likewise, the last column represents the consonants but without a trailing vowel. These "syllables" are of course just consonants, like the lone vowels we will still refer to them as syllables though we are aware that they are special cases.

We've glossed over the intersection of the last row and last column, "●". It is the vowel 'ɪ' which we do not use in English. This is not necessarily a defect; syllabaries may contain elements that ultimately are never used by a language and are just a part of the writing system for the logical completeness. More than one language may use a given syllabary and indeed make use of what we thought were dormant letters. Languages may not use a syllabary in identical ways and so we can expect that the vowel and (more likely) the consonant assignments to change a little. Hence, we will need to consider a user's locale when working on syllabic text in software.

In our simple syllabary the intersection of each row and column represents a unique syllable. Every row represents a syllabic series or "family" where all syllables begin with the same initial consonant and share a common graphic shape. Not all syllabaries are so regular that a series will make use of an underlying glyph, but the leading consonant must remain fixed. Some syllabaries will in fact give formal names to the syllabic families as well as the syllabic forms (the columns).

Objectives in Syllabic Pattern Matching

In essence, what syllabaries need from regex languages is the facility to specify any row or column within a matrix. For example, we would need a way to express "match only syllables in the form of 'a' ". Conversely, we would also want to specify somehow "match only syllables in the family of 'm' ". This is not a problem that alphabetic scripts face where the "C" and "V" components always have independent character codes. The fusion of the two is the crux of our problem as linguistic and orthographic interactions demand the we separate the two. The short coming of regular expressions languages that do not make this separation for us becomes a major limitation on their utility. To harness the full power of regular expression

systems developers generally have to first transliterate text into ASCII, perform pattern matches, and re-transliterate the results back into the starting script. The rest of this article addresses how this limitation might be overcome.

Syllabic Equivalence

One way to think of a syllabary is as if the different forms were all different "cases" of the same basic letter. This is useful to a point; we do want the capability to "fold cases" for case insensitive pattern matches. Of course, we realize this is an over simplification since each "case" is embedded with more linguistic information than orthographic case differences are ladened with.

The equivalence class model of POSIX style regular expressions is appropriate here. Equivalence classes essentially strip away diacritical, accent or tonal marks to match the underlying character. This is precisely what we need to accomplish for equating syllables in the same family. For instance, to match any member of the "▽" family we would construct an equivalence expression "`[#▽#]`". We can illustrate these results in our next table.

[#▽#]						
	a	**e**	**i**	**o**	**u**	+
d	□	◧	◨	⬒	⬓	■
l	△	◭	◮	◬	◬	▲
m	▽	▼	▼	▼	▼	▼
n	◇	◆	◆	◆	◆	◆
+	○	◐	◑	◓	◒	●

We need not define an equivalence for every member of the "▽" family. That is `[#▼#]`, `[#▼#]`, `[#▼#]`, etc. are not necessary. A syllabary can generally be expected to have nominated one form as a geometric base (the parent), the others are then considered to have been derived from that base. This may not be the case in syllabaries, such as Cherokee, that do not exhibit some degree of geometric regularity. Nonetheless, only one representative family member is required to define an equivalence class for the entire family.

Example:

> The transcription of the name "Emanuel" in our simple syllabary under Semitic rules would take a few forms:
>
> ○▽◆◐▲ (correct)
> ○▽◆◐▲
> ○▽◆◐▲
> ○▽◆▲
>
> To account for phonetic confusion, the above also repeats with ○ as ◐. Our expression to match all cases is then:
>
> [○◐]▽[#◆#](◐)?▲

Some languages will use a syllabary with redundant syllabic series (Amharic's use of Ethiopic is a good example) such that the "□" family might instead be a duplicate of the "◇" family which may happen when the language in question did not have the "n" sound in its phonemic repertoire. The consequence is that our `[#◇#]` equivalence would then include all "□" as well under a locale setting for such a language.

Syllabic Classes

Similarly, we will also want to detect an arbitrary syllable ending in a given form. Such is often the case when inflection occurs to show possession, pluralization, change of gender or article form of a noun. To match any syllable ending in the "i" form, for instance, we might employ the lone vowel "◑" to construct the class `[:◑:]`.

While understood, this approach is not ideal since it assumes the lone vowels of every script be defined as classes. Doing so would offer a way to restrict the script that the class applies to, but we can assume also that mixed script text is fairly uncommon. We can generalize the approach for all syllabaries by utilizing the notation of a syllable's ordinal form value (the column number) as per `[#4#]`. We count from 1 in our notation simply because syllabaries refer to the initial form as the "first" and never the "zeroth". An example is shown in our next table:

[#4#]						
	a	**e**	**i**	**o**	**u**	+
d	□	◧	◨	⬒	⬓	■
l	△	◭	◮	⧋	⧊	▲
m	▽	◸	◿	⧍	⧎	▼
n	◇	⬖	⬗	⬘	⬙	◆
+	○	◐	◑	◓	◒	●

Until now the choice of "#" for the syllabic notation has not been explained. Following experimentation with alternatives, the symbol has been chosen since it resembles the grid that syllabaries are invariably presented in and so provides a useful mnemonic device. In effect we use the symbol to indicate that we want to match the whole of either a row or column of a given syllabary.

Pattern transliterations and substitutions analogous to changing specified "cases" could also be performed by employing the syllabic class notation:

```
tr/[#1-3#]/[#4-6#]/;
```

and

```
$word =~ s/[#7#]([#□#])/[#2#]$1/g;
```

Syllabic Matching Constraint

The real power of the syllabic restraint notation comes out when we want to specify ranges of acceptable forms. To demonstrate range matching consider the pattern "`[△▽]{#3,4#}`", the expected matches are indicated in our next table:

[△▽]{#3,4#}	a	e	i	o	u	+
d	□	◧	◨	⬒	⬓	■
l	△	◭	◭	◬	◬	▲
m	▽	◸	◸	◺	◺	▼
n	◇	⬖	⬗	⬘	⬙	◆
+	○	◐	◑	◓	◒	●

Repeating with the slightly more complex pattern "`[□▽-◇]{#2,4-6#}`":

[□▽-◇]{#2,4-6#}	a	e	i	o	u	+
d	□	◧	◨	⬒	⬓	■
l	△	◭	◭	◬	◬	▲
m	▽	◸	◸	◺	◺	▼
n	◇	⬖	⬗	⬘	⬙	◆
+	○	◐	◑	◓	◒	●

Equivalently we could have expressed the pattern in the negative form as: "`[□▽-◇]{^#1,3#}`":

Further Reading

The goal of this essay was to present the case for syllabic character classes and pattern matching utilities by working with an abstract and very simple syllabary. Recall that our simple syllabary is a special case of an open syllabary, sufficient for laying the ground work for later working with full CVC syllabaries and syllabaries applying tones such as Yi. Working with a true syllabary and the localized rules that apply to it brings out more subtle levels of complexity that we have only touched on. Thoughts on applying the approach to syllabic regular expressions applied to Ethiopic script is available here.