Developing Amharic Regular Expressions in Perl

በዳንኤል፡ያዕቆብ

This document has been prepared for members of the `perl-unicode` email list to describe by way of example the limitations of working with the Ethiopic script under the `UTF-8` support in `Perl`. It is the intention that this document provide the essential information for regular expression library developers to add support for syllabic scripts to meet the needs and expectations of the user community.

Ethiopic is a syllabary whereby, unlike Hebrew and Malayalam, diacritic marks are *not* used to indicate the vowel element in the `CV` pair of the syllable. This is not to say that Ethiopic syllables are graphically irregular in the nature of Katakana, Hiragana, or Cherokee, obviously not. Unlike some syllabaries diacritic marks in Ethiopic are not graphically detached symbols, they are not independently encoded.

This indeed is the heart of the problem at hand. As diacritic marks are not independently encoded we must extract the vowel (or diphthong) component of the syllable by analysis of the character code (usually a modulo test). Such information would be definitive for the syllable's character class. However programming languages do not yet have support to detect syllabic classes. Nor do we have a facility in regex languages to specify the syllabic class of a character which is essential in linguistic and orthographic processing.

Lets consider now the Amharic simple plural and first person singular suffixes as an example of this second problem in particular. Most any study of Ethiopian languages applying the Ethiopic syllabary will present possible formations with these articles in a form similar to:

<noun> -o**ች** -e

which may also be represented with the `cv` *sequence:*

<(cv)(cv)(c)> -vc -v

The morphemes here are -o**ች** and -e and the sequence is sufficient as a regular expression for linguists. Unfortunately the orthography does not follow the morphology which is the onset to complexity. There is no way to write in Ethiopic the vowel element of a syllable which is why IPA symbols tend to be mixed with Ethiopic elements in language studies. A consonant in Ethiopic script on the other hand is generally considered to be the 6^{th} (counting from 1) syllable ("**ች**" in the above for example).

Using the above sequence and the example common noun for "house", "**ቤት**", we would obtain the following derivations:

ቤት	"House"
ቤቴ	"My House"
ቤቶች	"Houses"
ቤቶቼ	"My Houses"

This is indeed a very simple example, as there are 13 possessive forms that can follow the base noun, 9 of which can also follow the 6 pluralizing articles of Amharic common nouns. Valid combinations of the full collection of Amharic prefixes and suffixes can lead to derivations in the thousands. Fortunately it is all very regular and predictable. Only the logical expression in native script in a form following our linguistic understanding proves difficult.

Transliteration

Now we will build a regular expression for our example using Roman script. Transliteration into Roman script provides a work-around to the phoneme-morpheme boundary problem where the `cv` pairs of the syllables are split up and discretely known. We define then our regular expression in Perl with the aid of a few useful strings:

```
$plural = "oc";       # aka  -oች
$pos    = "e";        # aka  -e
$stem   = "bet";

/\A$stem($plural)?($pos)?\z/
```

Our regular expression expands into the expected valid formations:

Transliterated Regex		Resultant		Retransliterated Resultant	English Meaning
bet	⇒	bet	⇒	ቤት	"House"
bet(oc)	⇒	betoc	⇒	ቤቶች	"Houses"
bet(e)	⇒	bete	⇒	ቤቴ	"My House"
bet(oc)(e)	⇒	betoce	⇒	ቤቶቼ	"My Houses"

The primary advantage to using transliteration in regular expressions is that we never have to be concerned with handling the recombination problem of the consonant and vowel across the subexpression boundaries[1]. What we loose primarily then is the practicality of working in native script and the economic cost, in computing terms, of the transliteration and retransliteration work.

Faking Character Classes

Given the `UTF-8` support in newer versions of `Perl` we would hope to be able to be to leave transliteration behind. The Ethiopic programmer would most certainly be eager to do so and attempting to construct a regular expression for our four derivations would be lead most directly to:

[1] Across all grouping, character class and alternation boundaries.

```
/\Aቤ([ትቴ]|(ቶ[ችቼ]))\z/
```

Unfortunately it was required that we `chop` off the "ት" in our `$stem` "ቤት" and build it into the groupings and character classes of our regular expression. We have lost the convenience of our `$plural` and `$pos` abstractions and had to expand them into our subexpressions as well.

This is of course all overly specific to our example word "ቤት" or at least could be not be readily applied to words ending in other than "ት". As suggested in the 1997 paper[2] macros could be defined to help generalize our expressions:

```
$HAMS  =  [ሄሌሔሜሤሬሴሼቄቔቤቬቴቼኄኔኜኤኬኼዌዔዜዤዬዴዼጄጌጜጤጬጴጼፄፌፔ];
$SADS  =  [ህልሕምሥርስሽቅቕብቭትችኅንኝእክኽውዝዥይድዽጅግጝጥጭጵጽፅፍፕ];
$SABI  =  [ሆሎሖሞሦሮሶሾቆቖቦቮቶቾኆኖኞኦኮኾዎዖዞዦዮዶዾጆጎጞጦጮጶጾፆፎፖ];
```

For an arbitrary stem we may then construct:

```
my ($lastChar) = chop ($stem);
if ( /\A$stem($SADS|$HAMS|($SABI[ችቼ]))\z/ ) {
  ⋮
}
$stem .= $lastChar;
```

Test of the form:

```
if ( $word =~ /($SADS|$HAMS|($SABI[ችቼ]))\z/ ) { ...
```

are a bit better suited.

Defining Named Properties

We may still find the above approach to be inefficient and increasingly memory consuming and expressively ungainly as we would ultimately extend our regular expression beyond our simple singular instances of our example suffixes.

With recent development versions of `Perl`[3] we are given the capability to define new character properties through the newly introduced `\p` escape. The author has done exactly this in the package found at:

`ftp://ftp.cs.indiana.edu/pub/fidel/perl-unicode/perl-unicode.tar.gz`.

The package adds the character property definitions for Ethiopic that at this time have not made it into the pre-Unicode 3.0 based *UnicodeData* file. The package adds the new property type Y_i to identify the sYllabic order i of a syllable.

Ultimately however without the ability to check the character property values of the terminal characters in preceding subexpressions we return to the same expressive limitations we encountered with the faked character classes:

```
if ( $word =~ /[\p{Y5}\p{Y4}]|(\p{Y6}[ችቼ]))\z/ ) { ...
```

[2] "Thoughts on Regular Expressions for Ethiopic" `ftp://ftp.cs.indiana.edu/pub/fidel/perl-unicode/regex979.ps.gz`

[3] `Perl` versions 5.005_52 and later.

Back to the % Operator

In the 1997 paper the % operator was proposed for matching the syllabic class (vowel or diphthong property) of any preceding adjacent character that has a syllabic context[4]. The syllabary operator still seems to offer a certain convenience:

```
$plural = "%6ች";   # aka  -oች
$pos    = "%4";    # aka  -e

$stem   = "ቤት";    # we stick with our example though the $stem
                   # can now be arbitrary

/\A$stem($plural)?($pos)?\z/  # Same as the transliterated regex!
```

Our regular expression expands into the expected valid formations:

Regex		Expansion		Resultant	English Meaning
ቤት	⇒	*none*	⇒	ቤት	"House"
ቤት(%6ች)	⇒	ቤ(ት%6)ች	⇒	ቤቶች	"Houses"
ቤት(%4)	⇒	ቤ(ት%4)	⇒	ቤቴ	"My House"
ቤት(%6ች)(%4)	⇒	ቤ(ት%6)(ች%4)	⇒	ቤቶቼ	"My Houses"

The above expansions demonstrates what is truly desired in this field of text processing. The capability to compose general vs specific (expanded subexpressions) regular expressions in native script is also fundamental to being able to utilize the regular expressions language. Another generalized expression:

```
$prefix = "[በከየ]";
$plural = "%6ች";        # aka  -oች
$pos    = "%4";         # aka  -e

if ( $word   =~ /\A($prefix)?(\w{2,5})(($plural)?($pos))?\z/ ) {
  $stem =  $2;
    ⋮
}
```

The use of the syllabary class operator offers the convenience of constructing a regular expression in a form not far removed from how the orthographic morphology is understood. The apparent necessity of the operator would diminish were an escape construct capable of performing the same service. The author invites comments and suggestions to this end.

[4] Essentially % would work like (?<=\p{Yᵢ}) across preceeding subexpressions.