PToleMy
Transliterating Proper-Names

J D Riding
Linguistic Computing at
British & Foreign Bible Society

Abstract:

Transliterating proper-names accurately, consistently and acceptably across the full corpus of a bible, the translation of which may well take up to fifteen years to complete, is more difficult a task than might at first be imagined. A number of dynamics come into play including how the original name in the source texts is thought to have been spoken, how the name is spoken in any model text being used as a general source for the translation and, most importantly, how well this phoneme stream can be reproduced by speakers of the target language. Even having taken all this into account there are always some cases where other considerations require a rendering which conflicts with the generally accepted principles for that language.

Nevertheless, the majority of names can and should be rendered in the target language using a transliteration schema which is both acceptable to the reader and generally consistent throughout the text. This paper proposes a transliteration machine which can learn from early examples the most common transliterations for components of names. This information is then used to propose transliterations for names as yet unseen by the machine providing consistent renderings for names based upon the transformations learned.

1. Rendering proper-names

It must be acknowledged that no matter how much effort is put into identifying and applying a consistent transliteration schema, there will be still be circumstances in which the translators will, rightly, disregard it in favour of other more important considerations. For example, the translators of the Swahili Union Bible [4] transliterated the name Abram into Kiswahili as 'Abramu' adding the final 'u' out of deference to Kiswahili's preference for open syllables. This is entirely consistent with the expected transformation from English to Swahili. When, however, they came to the name Abraham, their choice was 'Ibrahimu', rather than 'Abrahamu' as a table of consistent transliteration might have predicted. In this case the influence of Arabic in

coastal East Africa was so strong that it was felt appropriate to conform to the local tradition as derived from Koran. Such considerations will always require translators to disregard a general schema of transliteration in favour of the a more culturally acceptable alternative. Setting aside such considerations the remainder of the proper-name set can be treated systematically.

There are a number of issues to be considered when deciding how best to render a proper-name in transliteration. In addition to the question of different scripts the translator must also consider whether the sounds which represent that name in the source and/or model texts can be reproduced easily by speakers of the target language and if so how best to render them in the target language script. All of these considerations contribute to the complexity of developing a consistent transliteration schema.[1]

2. What's in a name?

For the purposes of this discussion we shall consider a name as an utterance stream comprised of individual phonemes [4:23]. We expect some mutation to occur dependent upon the limitations of the target language phoneme set which may make certain juxtapositions of phonemes difficult for speakers. We also expect phonemes to map to one or more characters in the target language script and we recognise that there may be more than one solution to the question of how to render a particular phoneme or phoneme cluster.

In an ideal world we would have access to a table of phonemes for every known language and the task for a transliteration machine would be no more than loading the appropriate map and generating renderings from it. In practice this scenario is unlikely. We can always ask the translators to create such a map but ideally we would prefer to be able to ask the machine to discover for itself the various components. In practice we find that the individual components we want to transliterate tend to correspond most closely to the atoms of syllables [4:230]. These are generally categorised as the components of syllable onset and rime, with rime further sub-divided into nucleus and coda. These components are generally synonymous with phonemes although they may also represent phoneme clusters whether vocalic or consonantal. Some languages, particularly tonal languages, have the concept of *toneme* or *chroneme* but for the purposes of this task these can be considered as largely synonymous with phonemes.

1 For a general discussion of proper-names in scripture see [1].

2.1 finding syllables automatically

Many systems exist to identify syllables in natural language. Typically they are language specific and often make use of dictionary lookup tables to resolve ambiguities. Given that most Bible Translation takes place in developing world vernacular languages such aids are rarely available nor is it the case that the necessary skills will always be available to create them. A transliteration machine must, therefore be able to discover for itself, with a reasonable degree of accuracy, syllable boundaries within a word. This at once raises another problem in the context of proper-names. Since the source (and perhaps model) languages for biblical names are likely to be remote to the target language it may well be the case that the syllable boundaries occurring naturally in the target language will not always agree with those proposed by a name. For example, In English 'Abraham' is generally thought of as containing three syllables: A:bra:ham. This is in conflict with the Hebrew original where the syllable boundaries fall אב:ר:הם. How much this matters is very much an open question. In the case of a target language which is linguistically fairly close to the source language there might be value in reproducing the syllable structure of the original but in most cases this seems unlikely. The consequence of such ambiguity is that an analysis of the word list in general (without proper-names) may identify a syllable structure which is in conflict with that found in the proper-name list. Having noted these caveats it ought still to be the case that a reasonable attempt can be made for most languages to identify syllables and from them their components. These components, onsets, nuclei and codae, then become the building blocks for a transliteration schema.

At present Paratext has the ability to identify to some degree syllable boundaries in languages using Latin script. This ability is dependent upon hard-coded rules identifying vowels and other nuclei and noting illegal letter combinations. For a truly language independent solution we prefer making the attempt to identify automatically these elements. This will not always be possible, languages such as Czech where nuclei are not always written as in Plzeň may well require a nucleus table to be created by the user. Many languages, however, will allow such information to be generated automatically [1,5].

Once a component table has been constructed names can be broken down into their component parts. These components can then in turn be mapped between the source/model and target languages. Such an analysis of the English name Joshua would mark syllable boundaries Jo:shu:a and syllable com-

ponent boundaries J·o:sh·u:a. A similar analysis of the Kiswahili names list would produce Y·o:sh·u:a. In this simple example we are left with five ordered components in each case which can be mapped unambiguously thus:

{(J=Y) , (o=o) , (sh=sh) , (u=u) , (a=a)}

Other names will be less obliging but this example has served to demonstrate the principle. Where names do not break down into conveniently aligned components more work must be done to identify the corresponding elements in each name. If we return to our first example 'Abraham' we find that mapping this from English to the corresponding phonemes in Tonga is rather more dificult. Tonga has no 'r' phoneme, preferring 'l'. Worse still, the 'bl' letter pair cannot be used without the insertion of a vocalising 'u' thus: 'bul'. The consequence of this is that the analysis for the English A:br·a:h·a·m must be mapped to A:b·u:l·a:h·a:m·u. The outcome we are hoping for from this would be:

{(A=A) , (br=bul) , (a=a) , (h=h) , (a,a) , (m=m) , (_=u)}.

or better still:

{(A=A) , (b=bu) , (r=l) , (a=a) , (h=h) , (a,a) , (m=m) , (_=u)}.

Persuading a machine to recognise these complexities and identify a non-concatenative sequence of matched items, without recourse to supplied tables, is not trivial.

2.2 Aligning disparate datasets

BFBS LC has a long term research project looking at exactly this problem of aligning sets of data where not all the members of each set will map successfully or consecutively to one another. The approach adopted by BFBS is based on an algorithm first developed to identify sub-atomic particles created momentarily in the laboratory. Such particles are the outcome of collisions between atoms, in effect fragments of the original atoms. They exist only for an instant, they cannot be seen directly but their moment of existence can be traced on a screen, much like footprints in a snow field. The difficulty is that any given collision may generate an unknown number of particles and all appear (and disappear) at the same instant. The number and type of particles produced can only be discerned from the 'footprints' they leave behind.[2] Deciding which of the footprints form a track made by a single particle is analogous to the problem of deciding which components in a phoneme sequence are related. At BFBS we characterise this problem as a simple two-dimen-

2 For an accessible example of sub-atomic particle tracking see [3].

sional matrix with one potentially related item along the x axis and one along the y axis. For ease of description we shall work this example using just alphabetic characters, and sequences which share the same script.[3]

Ø	a	b	u	l	a	h	a	m	u
a	**a**				**a**		**a**		
b		**b**							
r									
a	**a**				**a**		**a**		
h						**h**			
a	**a**				**a**		**a**		
m								**m**	

fig 1

Where items can be matched in the two streams the intersection of the match is noted as above. To the human being, identifying the most productive matches is fairly straightforward. The machine must be given strategies to identify the best matches. The best solution to the matches will be the one that connects the maximum number of intersects across the graph such that the items linked remain unique, ordered and consecutive within their original streams. Where more than one solution of identical length is generated the solution preferred should be that within which the matches occur in closest proximity to one another in both streams. To illustrate this we shall work the first few preferred matches in the example above:

Each intersect is represented by the letter matched in the x stream and its index number in the stream, followed by the letter matched in the y stream and its index number in that stream. The set of intersects generated by the matrix above is:

$$\{ (a_1,a_1),(a_1,a_4),(a_1,a_6), (b_2,b_2), (a_5,a_1), (a_5,a_4),$$
$$(a_5,a_6), (h_6,h_5), (a_7,a_1), (a_7,a_4), (a_7,a_6), (m_8,m_7) \}$$

The human being sees at once that the best match is likely to fall diagonally across the centre of the matrix, the machine knows no such thing. To the machine the first valid intersect could be any one of the first three items in the intersect set i.e. (a_1,a_1), (a_1,a_4) or (a_1,a_6). The machine must calculate for each one of these the next valid matching pair of letters and so on across the matrix. Taking each of these in turn we can calculate which if the other inter-

3 Where sequences are drawn from languages with different scripts an additional layer of information mapping the equivalent characters in the two scripts is required.

sects can follow. For a following intersect to valid both the indices of the following match must be greater than corresponding indices of the first match. Using this rule we can identify the following valid followers taking in each case the follower closest to the first intersect:

$$(a_1,a_1), (b_2,b_2)$$

$$(a_1,a_4), (h_6,h_5)$$

$$(a_1,a_6), (m_8,m_7)$$

We then repeat this for the second intersect and so on across the matrix. This generates the following series of potential match sets:

$$(a_1,a_1), (b_2,b_2), (a_5,a_4)$$

$$(a_1,a_1), (b_2,b_2), (a_5,a_4), (h_6,h_5), (a_7,a_6), (m_8,m_7)$$

$$(a_1,a_1), (b_2,b_2), (a_5,a_6)$$

$$(a_1,a_1), (b_2,b_2), (a_5,a_6), (m_8,m_7)$$

$$(a_5,a_1), (h_6,h_5), (a_7,a_6), (m_8,m_7)$$

$$(a_7,a_1), (m_8,m_7)$$

Clearly the second option aligns the largest number of intersects from which we conclude that *r≡ ul* since these characters represent the complement of the matched item set. Recognising that, as discussed above, *br≡bul* requires the identification of '*br*' in English as a valid onset cluster. Similarly, syllable analysis tells us that Tonga does not permit closed syllables allowing us to hypothesise a voicing after the final English '*m*'. If we apply this knowledge to the problem we can redraw the matrix thus:

Ø	a	b	u	l	a	h	a	m	u
a	**a**				**a**		**a**		
br									
a	**a**				**a**		**a**		
h						**h**			
a	**a**				**a**		**a**		
m								**m**	
_									

At first sight this may seem a less good result. Our intersect set now looks like this:

$$\{(a_1,a_1), (a_1,a_3), (a_1,a_5), (a_5,a_1), (a_5,a_3), (a_5,a_5),$$
$$(h_6,h_4), (a_7,a_1), (a_7,a_3), (a_7,a_5), (m_8,m_6)\}$$

From which we can derive match sequences:

$$(a_1,a_1), (a_5,a_3), (h_6,h_4), (a_7,a_5), (m_8,m_6)$$

$$(a_1,a_3), (h_6,h_4), (a_7,a_5), (m_8,m_6)$$

$$(a_1,a_3), (a_5,a_5), (m_8,m_6)$$

$$(a_1,a_5), (m_8,m_6)$$

$$(a_5,a_1), (h_6,h_4), (a_7,a_5), (m_8,m_6)$$

$$(a_7,a_1), (m_8,m_6)$$

As before we prefer the longest match sequence, in this case the first one. The complement of this sequence maps *br=bul* and *_=u*. This is a much more plausible analysis but it is dependent upon knowing, or being able to calculate, how syllables are formed in the source and target languages. We can also deduce from the fact the Tonga prefers open syllables the probability that *br/bul* should be mapped *b_r/bul* indicating a strong probability that *b=b* and *r=l*, correctly mapping the phoneme shift between the alveolar English '*r*' and the Tonga '*l*' [4:35].

At this stage, however, much of this is speculative but as more and more pairs of names are aligned and the results recorded we can expect the most common mapping to be reinforced. A bigger problem is deciding which of two or more match sequences of equal length is to be preferred.

2.3 Calculating the best match sequence

Where an analysis generates two possible match sequences of identical length we must find a means to assess which is more likely to be the best solution. We do this by favouring sequences and sub-sequences where the individual intersects are most proximate. We can calculate this as follows:

Let us consider our sequences, which we can call *X* and *Y,* as ordered n-tuples, each containing a row of intersect coordinates where each entry has a value and an index *i* in the row as indicated by our subscript indices. Let *MS* be an array of sets of matched characters which can be generated between *X* and *Y* following the rules we have laid down for valid sequencing. Let M_i be the *i*-th match array in *MS*. Let p_n represent an element from M_i where *p* is an ordered pair of matching characters, the first character matched from the *X* sequence, the second from the *Y* sequence. Each item in the pair carries a subscript index showing its position in its respective n-tuple.

In our analysis above we generated a set of match sequences in which two

pairs of sequences shared the same length. Without some means to identify the stronger of the two sequences in each pair we must value them equally since both are the same length. For clarity we shall select the shorter pair:

$$(a_1,a_5),\ (m_8,m_6)$$

$$(a_7,a_1),\ (m_8,m_6)$$

We need a way to assess which of these sequences represents the strongest relationship. We shall base our decision on the proximity of each matched characters with the following matched character in their respective streams. This can be expressed as the greater of the distances between the two characters as they are found in the original two n-tuples. In the first example the greater distance *d* between the two matching items is *8 − 1 = 7*. We convert this into a useful value as follows:

$$1+(1-\frac{d}{f})$$

where *f* represents the maximum distance between two pairs of matching characters at or below which it is reasonable to hypothesise a relationship. By default *f* is set to the value 10. This is adequate for most natural language although it is possible that some highly-agglutinative languages which exhibit average word lengths significantly greater than 20 letters might require a higher value for *f*. In this case *f = 10* gives a relationship value of *1.3*. We calculate this value between each ordered pair in M_i and then calculate a value for M_i as a whole as the product of these individual values:

$$\prod_{pn=1}^{pn=|M_i|-1} 1+(1-\frac{d}{f})$$

where p_n and p_{n+1} represent each adjacent pair of ordered pairs in M_i in turn. In the case of our two item example above the value remains *1.3* but given a longer match string such as $(a_1,a_1),\ (a_5,a_3),\ (h_6,h_4),\ (a_7,a_5),\ (m_8,m_6)$ we are now able to evaluate the sequence as a whole, finding that it returns a relationship value of *10.97 (1.6 • 1.9 • 1.9 • 1.9)*. The same calculation for the pair of two item sequences returns, as we have seen, *1.3* for $(a_1,a_5),\ (m_8,m_6)$ and *1.5* for $(a_7,a_1),\ (m_8,m_6)$. Thus, there is no question which is the best sequence but if we were faced with choosing one of a pair of sequences with the same length we should favour $(a_7,a_1),\ (m_8,m_6)$.

Thus far our discussion has been largely analytical as we seek to provide the machine with the ability to align the two phoneme streams and learn the mappings between them. If we are to build a machine which is truly language independent this analysis is a pre-requisite as we simply cannot as-

sume that the information will be available from other sources. We now move on to consider how we can use this analysis in a generative capacity to hypothesise valid and consistent transliterations for other names.

3. Recording phoneme maps

Once we have generated a plausible map, or perhaps set of maps, marking the equivalent phonemes between any pair of names we need to add the results to a database of phoneme maps. We build this map as a simple two dimensional matrix recording the relationships discovered between the names in the two languages. Such a map for English and Greek would look something like this:

	a	b	c	ch	d	e	f	g	h	i	j	k	l	m	n	o	p	ph	ps	q	r	s	t	th	u	v	w	x	y	z
α	9								3																					
β		9																												
γ								6																					3	
δ					9																									
ε						9																								
ζ																														6
η	4					5																								
θ																								9						
ι										9																				
κ			2									8																		
λ													9																	
μ														9																
ν															9															
ξ																												9		
ο																6														
π																	9													
ρ																					9									
σ																						9								
τ																							9							
υ																									9					
φ							3											9												
χ				9																										
ψ																			9											
ω																6														

fig 2

Here the most likely matches for individual phonemes have been calculated and a value entered at each possible match point representing the likelihood that these two phonemes would transliterate in this way. The result is a table which is both analytic and generative. The data contained within it represent the analysis of how the phoneme sets in these two languages can be mapped. Once the table has been built we can then use it to generate transliteration hypotheses for names as yet unseen simply by locating the phonemes in a name on one side of the table and reading off the possible transliterations on the other. As the user validates or corrects these hypotheses the results can be recorded in the table via a feedback loop which will allow the system to continue to refine its hypotheses.

At BFBS this transliteration system is known as PToleMy – Proper-name Transliteration Matrix (PTM). Early versions of PToleMy were developed in the early 1990s but since then there has been no opportunity to update the process for Unicode. Building a PTM is clearly something that can be done *per manu* but much of the process can be automated. Once the items in each language which represent phonemes have been identified each set is ranged along one of the axes of the matrix. The matrix can now be trained to identify the equivalent items in the two sets automatically by providing a set of training data. In this context the best training data is a list of pairs of proper names from the source and target texts. The names must be the same length and their phonemes aligned. All cells in the matrix are set to 0.[4] Each phoneme from each pair of names is now taken in turn and the value of the intersection between the two phonemes is incremented. When this has been done for all the training set the matrix can then be used to hypothesise transliterations based upon the loadings set by the training data.

4. Next Steps

This is very much a work in progress. At present it is not clear which of the approaches detailed above will prove most helpful. PToleMy development is currently in hand at BFBS in conjunction with the University of Aberystwyth who are supplying most welcome support in the form of a research assistant. The initial phase of the project is the development of a system to build a PTM from a supplied set of training data. Once this is in place we hope to address the more complex issue of automatically aligning name pairs as described in 2.2 above.

4 At this point any existing information about phoneme equivalence can also be stored as values within the matrix.

Longer term proposals include developing multi-dimensional transformations to allow other information such as preferred vowel and consonant shift to be pre-loaded as an additional layer and the use of IPA as an intermediate stage of processing.

J D Riding, BFBS LC
January 2011

Bibliography

[1] Bailey, N. (2007), Proper Names in the Bible: translation and transliteration issues, *Word & Deed*. SIL.

[2] Black, H. A. (1999), *Requirements for a Syllable Parser*, Technical report, SIL.

[3] Kirke, A. (2010), *Cloud Chamber – A live radioactive music duet with violinist and subatomic particles.* www, retrieved 8th Dec 2010. http://cmr.soc.plymouth.ac.uk/alexiskirke/CloudChamberInfoSheetWeb. pdf

[4] Ladefoged, P. (2001), *A Course in Phonetics*, Harcourt.

[5] Riding, J. D. (2005), *First Experiments in Automatic Hyphenation*, Technical report, British & Foreign Bible Society.

[6] UBS, ed. (1989), *The Holy Bible in Kiswahili, Union Version*, United Bible Societies.