What's in a Name?

Jon D Riding
United Bible Societies
Stonehill Green, Westlea,
Swindon SN5 7TJ
jonriding@biblesocieties.org

Neil J Boulton
United Bible Societies
Stonehill Green, Westlea,
Swindon SN5 7TJ
neilboulton@biblesocieties.org

Abstract

This paper describes the development of a language independent process for identifying proper-names in a text. The process is derived from a machine originally designed to analyse non-concatenative morphologies in natural languages. The particular context for this work is the task of managing the 5,000 or so proper-names found in a Bible, including the identification of close cognates and reporting instances where a related form does not appear to be present.
The need for such a system is explained and the process by which the machine is able to identify names in the target text is described. The problems posed by disparate orthographies are noted. Results obtained from Eurasian, South American and African languages are discussed, common problems for the process identified and its possible use in the context of technical vocabulary suggested. Commonalities between the task of identifying morphology templates, ordered phoneme sets and syntax patterns are noted.

1 Introduction

The Bible translator faces a peculiar set of problems. Some are common to all translation but others are particular to the task of translating a Bible. In this paper we shall explore a problem particular although not unique to Bible translation and describe how NLP techniques might be employed to limit the scale of the difficulties it causes for translators. The issue is that of managing the transliteration of proper-names found in the Bible.
At first sight this might seem a strange focus for NLP research but there are some characteristics of a Bible translation project which make it far from trivial. To begin with, the Bible is a large book. In its largest form (not all Christian denominations read all of it) it comprises 81 books whose origins can be traced back something like 3,500 years. Some are histories, others are collections of poetry or wise sayings, some are telling critiques of the politics of the times and others record personal and formal correspondence between churches and friends. As a collection it is vast and eclectic. All of these genres have in common the use of narrative to expound their message and all good stories are, in the end, about people. All of these people, their nations, tribes, cities and families have names.
The Bible contains references to between 4,500 and 5,000 names. The precise number depends on whether categories such as tribal names, names of regions etc... are included. All these names are 2,000 years old or more and as such they arise from cultures where a name was of particular significance. Not only did a name seek to express character it could also be regarded as formative of character [Blumenthal, 2009]. Biblical names, therefore, add both dimension and depth to a narrative [Krasovec, 2010]. Given this, it becomes very important that they are handled consistently by a translation team.
The translation team also brings its own problems. A translation of the Bible rarely completes in fewer than ten years. Fifteen years or more is not uncommon. Over such a period the membership of the team may well change and the team's expertise will develop as the project progresses. For all these reasons and more it is not hard to imagine the importance of handling

Proceedings of the 38th Conference Translating and the Computer, pages 122–132,
London, UK, November 17-18, 2016.

names consistently and the difficulties a team may encounter in ensuring that this is applied throughout the text.
Unlike the bulk of a text, proper-names are rarely translated [Auden, 1970], it is more usual that they are transliterated[1]. Transliteration involves recasting the phoneme stream which represents the model name in the base text into an appropriate form for the target language. Sometimes entire translations undergo this process when more than one orthography is used for a language (as in Romanian/Moldovan prior to 1989 [Chinn, 1993]). The transliteration task is the same for any given pair of model and target languages, to find a way of representing in the target language an ordered set of phonemes from the model language. Other changes may also take place where languages are more highly inflected. Our task has been to find a way of identifying the thousands of proper-names in a new translation of the Bible so that the translators can more easily review their work and ensure they have rendered names consistently throughout the text.

2 Components of Proper Names

Phonemes

Those of us accustomed to working with language as text tend to think of words as made up of letters. We recognise that particular surface forms are constructed from stem lemmata via morphological transformations but our day to day encounters with language as text encourage us to imagine that the fundamental building blocks are letters. This is both helpful and misleading. It is helpful insofar as letters map to typescript and so to the keyboards which are the medium through which we create text, it is misleading insofar as letters are only an approximation of the phonemic components which underpin spoken language. The system that truly underpins language is fundamentally about speech.
A proper-name (PN) is in reality a row of phonemes[2] which represent the sounds of that PN as an utterance. Many of these phonemes will correspond to letters in the alphabet but others are represented by letter clusters. English, for example, has a number of phonemes which cannot be represented by a single letter such as {*ch, ph, sh, th*} and it can be argued that some English vowels too may be represented by letter clusters e.g. {*ai, ea, ee, etc…*}, the list is incomplete. It is not our intention to argue the case for a defined and generally agreed phoneme set for English or any other language but rather to suggest that expanding our set of symbols beyond the alphabet to include common letter clusters which are used consistently to represent particular sounds is beneficial to the task of identifying PNs in text.

Order

If phonemes may be considered the building blocks of names there is a second element of equal importance. Order is critical in rendering and recognising names. Consider ‘David’. We might parse David into five components {D.a.v.i.d} but without taking into account that these components are ordered we get little benefit. ‘additive’, for instance, has all of the components found in ‘David’ but there is no relationship between the two words.

A PN then, is constructed from letters and letter clusters each of which may be taken to represent a phoneme and all of which are ordered to form the PN as a whole. A PN is, in other words, an ordered set of phoneme items.

---

1 There are occasional exceptions where names carry root meanings which are central to the narrative and translators decide to translate e.g. לֹא עַמִּי (Lo-Ammi) [Kittel, 1997] – Not-My-People [GNB, 1976] – Hos 1.9.

2 In this paper we use the word phoneme in a less than formal sense to describe an alphabetic character or cluster of characters that consistently represents a particular sound in a language. For a more formal discussion of phonemes see: [Davenport & Hannahs, 2010:115-132] or [Ladefoged, 2001:23-24].

3 Finding a name in text

Since PNs do not typically translate but are reproduced as a phonemic pattern in the translated text it follows that finding a PN in a text is an exercise in identifying the phonemic components of the PN ordered as we might expect them to be. Thus if we begin with a model representation of a name such as 'David' we might hope to find a similarly ordered set of items representing those sounds in another language. One of the early questions a Bible translation team must address is what they consider their 'model' to be for names. There is, rightly, an emphasis in Bible translation on returning to the Hebrew and Greek base texts but this is not always helpful. The reality on the ground may well be that the people group for whom the new translation is being prepared have had access to other translations of the Bible in regional *lingua franca*. Thus, for example, a Swahili translation may well choose to transliterate the name *Abraham* as *Ibrahimu*[3] rather than the more naturally Swahili *Abrahamu*. Such decisions recognise the need to work with existing expectations within the community rather than seeking to impose a purist view.

Where a team elects to transliterate directly from the base texts we must first render the model name into the same alphabet as the target text. This is done via a transliteration matrix which implements a simple Markov Chain[4] to record the probability of a particular base language phoneme being represented as a certain target language phoneme. Standard transliteration matrices exist for many pairs of scripts which can be adapted for particular language pairs. As the system identifies pairs of model and target names the individual transliterations these pairings represent are fed back into the matrix to improve subsequent transliterations.

Where our model and target texts share an alphabet we might hypothesise that *David* and *Daavidille* (Finnish, allative [Karlsson, 1999:119]) are references to the same PN by virtue of the two renderings sharing a common sequence of phonemes {D.a._.v.i.d_}. The order of these matched items is crucial to the identification as too is the proximity of each successive matched item to its predecessor. We should also note that we have preserved the leading capital in our examples so far but in practice many languages do not use leading capitals with PNs. In reality our matched set of phonemes needs to be {d.a._.v.i.d_} (we use '_' to represent lacunae covering one or more items).

The method adopted to assess how closely a word in a translation matches a model form of a PN is an adaptation of a process presented at TC34 [Riding, 2012]. The process compares candidates from the target text with a model form by ranging the model and candidate along the two axes of a simple matrix with the candidate on the *x* axis and the model on the *y* axis.

Thus in our example above, *David*/*Daavidille*, we generate the matrix:

		1	2	3	4	5	6	7	8	9	10	
	∅	d	a	a	v	i	d	i	l	l	e	
1	d	**d**					**d**					
2	a		**a**	**a**								
3	v				**v**							
4	i					**i**		**i**				
5	d	**d**					**d**					
												∅

Fig 1.

3 Swahili Union Bible [UBS, 1989]. In East Africa the influence of Arabic is strong, particularly closer to the Indian Ocean coast as a consequence of centuries of trade with Arab merchants.

4 For a full discussion of Markov Chains see [Puterman, 2005]

Taking origin at top left we now construct all the possible routes across the matrix via the matched items taking due account of order by following the rule that a successor must have x and y coordinates which are both greater than its predecessor. This results in six possible solutions:

S1 {d(1,1), a(2,2), v(4,3), i(5,4), d(6,5)}
S2 {d(1,1), a(2,2), v(4,3), i(7,4)}
S3 {d(1,1), a(3,2), v(4,3), i(5,4), d(6,5)}
S4 {d(1,1), a(2,2), v(4,3), i(7,4)}
S5 {d(6,1), i(7,4)}
S6 {d(1,5)}

All of these sequences are possible solutions to our problem. Clearly, we need a way to assess their relative merits. We do this by noting the degree of proximity for each pair of matched items. In S1 we have a set of four matched pairs:

{(d(1,1), a(2,2)), (a(2,2), v(4,3)), (v(4,3), i(5,4)), (i(5,4), d(6,5))}

Reading across the x coordinates we discover proximities for each of pair of 1, 2, 1 and 1. This measure of distance between each pair allows us to construct a score which represents the closeness with which the sequence matches the model. Given that we can calculate the value of a perfect match we express the value of this sequence as a probability with respect to a perfect match value. A more detailed description of the process is given at Appendix A.

4 In practice

This processing is now in beta test as part of the UBS CogNomen (CGN) system. CGN sits beside the translators' editor, ParaTExt, and reviews the new text as it is created. Tables exist which identify locations in the text where particular PNs might be present and using these CGN loads the model forms of those names from a model names list. Once a user has finished editing a verse CGN runs in the background and identifies candidates for each expected name storing them, together with their score, in a growing table of PNs for the new text. Names form part of a wider list of key terms which are carefully reviewed as a translation proceeds. At such reviews CGN provides a list of all names found in the pericopes under review, allowing the translator to approve renderings or not as they choose. Over time the translation team will identify a threshold value above which CGN's suggestions can be accepted with confidence without the need for detailed review.

As the translation approaches completion CGN's tables of PNs allow the team to focus on areas where review is needed, greatly reducing the scale of the task. Additional benefits are to segregate PNs from morphology based spelling checks in which context they represent little more than noise. It has been estimated that if CGN is used consistently throughout the lifetime of a project a saving of up to 5% of the time needed for the translation can be made.

5 Wider application

The application described here is particular to Bible translation but we believe that there are wider contexts for this processing. We know already that the process can automatically analyse complex discontinuous morphologies [Riding, 2012] and experiments suggest that progress might also be made in the analysis of syntax structures using a similar technique. Given that at word level this process represents a way to identify cognate forms which share a similar phonemic pattern we also hope to explore the automatic creation of technical indexes and glossaries.

6 Conclusion

The process described in this paper is able to analyse linguistic structures in a number of different and, at first sight, apparently unrelated contexts. That a single analysis process can be used in the context of phonemic and morphemic word formation, and possibly in syntax analysis, is telling us something significant about language. In every context this process relies upon identifying not only individual items within a text stream but also the order in which they are encountered. The moment we write something down we have removed it from the event stream of time. A document is not really a moment in time but a recording of a linguistic event stream conveying meaning not only in its individual components but by the order in which we encounter them.
CogNomen represents just one context in which stream based processing can help translators manage elements of large texts in an efficient and consistent manner.

Acknowledgements

We are indebted to United Bible Societies with whose support this work has been funded by Every Tribe and Every Nation (ETEN). Thanks are also due to Oxford Brookes University for continued access to their computing and library services.

References

Auden, W H (1970) A Certain World, The Viking Press, New York.
Bible Society eds. (1976) Good News Bible. BFBS
Blumenthal, F (2009) Biblical Onomastics: What's in a name? *Jewish Bible Quarterly Vol.* 37(2) 124-128
Chinn, J (1993) "The Politics of Language in Moldova", *Demokratizatsya* 1993 Vol. 2(2) pp. 309–315
Davenport, M and Hannahs, S J (2010) Introducing Phonetics and Phonology, 3rd Ed. Routledge.
Davidson, B (1981) The Analytical Hebrew and Chaldee Lexicon, 3rd Ed. Bagster, London.
Karlsson, F (1999) Finnish: An essential grammar, Routledge, London.
Kittel, R. (Ed.) (1997) Biblia Hebraica Stuttgartensia, Deutsche Bibel Gesellschaft, Stuttgart.
Krasovec, J (2010) The Transformation of Biblical Proper Names, T & T Clark International
Ladefoged P (2001), A Course in Phonetics, Harcourt, Florida
Puterman, M L (2005) Markov Decision Processes: Discrete Stochastic Dynamic Programing. *Wiley Series in Probability and Statistics*, John Wiley & Sons, New York.
Riding, J D (2012) Hunting the Snark - the problem posed for MT by complex, non-concatenative morphologies. In *Proceedings: Translating and the Computer 34,* ASLIB.
United Bible Societies eds. (1989) The Holy Bible in Kiswahili, Union Version, UBS

Appendix A: Processing Model

The Match Matrix

We begin by arranging the two items to be tested along the axes of a two dimensional matrix (fig 1). This example tests an English model *Abraham* against a Bantu candidate *Abulahamu*. Each word is bounded by the null set marker: ∅ with each cell numbered from 0 on each axis. Any individual character matches between the two names are marked by inserting the matched character into the cell at the intersection of that characters position in the two names. In this example the matched character 'b' is found in each word and the intersection of its position in each name is at coordinates (2,2) on the matrix. Likewise, the character 'h' is found in both names and the intersection of its position in the two names at at (5,6). 'm' similarly is marked at (7,8). The character 'a', however, appears three times in each name and so is matched at nine positions on the matrix:

{(1,1), (1.5), (1,7), (4,1), (4,5), (4,7), (6,1), (6,5), (6,7)}

Although we have identified all the individual character matches between the two names the machine needs a way to determine what might be the best sequence of these matches between the two names. The key to this is sequence. Character matches can only be considered valid if they are in sequence. For a match to be sequential to its predecessor in the sequence both the x and y coordinates of its match must be greater than those of its predecessor. Thus a(1,1) → b(2,2) is a valid sequence but a(1,1) ↛ a(1,5) is not as both matches share a common *x* index.

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	**a**			**a**		**a**		
2	b		**b**						
3	u								
4	l								
5	a	**a**			**a**		**a**		
6	h					**h**			
7	a	**a**			**a**		**a**		
8	m							**m**	
9	u								
10	∅								∅

fig. 1

n.b. As much of this appendix will be taken up with descriptions of match sequences we shall henceforward abandon the traditional syntax for listing *x,y* coordinates as it will soon become clumsy in favour of: c^x_y where 'c' represents the matched character, 'x' its x index and 'y' its y index.

Our rule of sequence serves also to prohibit match sequences where a successor is not the immediate successor. This whilst $b^2_2 \rightarrow h^5_6$ obeys the rule that both coordinates of the successor must be greater than the coordinates of the predecessor h^5_6 is not the immediate successor of b^2_2 and so is not truly sequential. The best sequence from b^2_2 is in fact a^4_5 which then leads us on to h^5_6. We have in fact found a three character sequence: $b^2_2 \rightarrow a^4_5 \rightarrow h^5_6$. Closer examination of the matrix shows us that this sequence is in fact part of a longer sequence beginning at a^1_1 (we disregard the null set boundary markers for clarity). Starting the sequence at a^1_1 gives us a match sequence across the whole matrix of: $\{a^1_1 \rightarrow b^2_2 \rightarrow a^4_5 \rightarrow h^5_6 \rightarrow a^6_7 \rightarrow m^7_8\}$. This looks like the best match sequence across the whole matrix but it is not the whole story. There are in fact seven possible match sequences across the matrix (disregarding sequences of cardinality < 2):

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	a			a		a		
2	b		b						
3	u								
4	l								
5	a	**a**			a		a		
6	h					h			
7	a	a			**a**		a		
8	m							**m**	
9	u								
10	∅								∅

fig. 2: $S_1\{a_5^1 \rightarrow a_7^4 \rightarrow m_8^7\}$

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	a			a		a		
2	b		b						
3	u								
4	l								
5	a	a			a		a		
6	h					h			
7	a	a			a		a		
8	m							m	
9	u								
10	∅								∅

fig. 3: $S_2\{a_5^1 \rightarrow h_6^5 \rightarrow a_7^6 \rightarrow m_8^7\}$

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	a			a		a		
2	b		b						
3	u								
4	l								
5	a	a			a		a		
6	h					h			
7	a	a			a		a		
8	m							m	
9	u								
10	∅								∅

fig. 4: $S_3\{a_1^1 \rightarrow b_2^2 \rightarrow a_7^4 \rightarrow m_8^7\}$

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	**a**			a		a		
2	b		**b**						
3	u								
4	l								
5	a	a			**a**		a		
6	h					**h**			
7	a	a			a		**a**		
8	m							**m**	
9	u								
10	∅								∅

fig. 5: $S_4\{a_1^1 \rightarrow b_2^2 \rightarrow a_5^4 \rightarrow h_6^5 \rightarrow a_7^6 \rightarrow m_8^7\}$

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	**a**			a		a		
2	b		**b**						
3	u								
4	l								
5	a	a			a		**a**		
6	h					h			
7	a	a			a		a		
8	m							**m**	
9	u								
10	∅								∅

fig. 6: $S_5\{a_1^1 \rightarrow b_2^2 \rightarrow a_5^6 \rightarrow m_8^7\}$

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	a			**a**		a		
2	b		b						
3	u								
4	l								
5	a	a			a		a		
6	h					**h**			
7	a	a			a		**a**		
8	m							**m**	
9	u								
10	∅								∅

fig. 7: $S_6\{a_1^4 \rightarrow h_6^5 \rightarrow a_7^6 \rightarrow m_8^7\}$

Given seven valid match sequences across the matrix (figs. 2-8) how is the machine to decide which is best? A trivial solution is to take the longest as the one likely to be the best but there are circumstances in which this may fail, for example:

	0	1	2	3	4	5	6	7	8
0	∅	a	b	r	a	h	a	m	∅
1	a	a			**a**		a		
2	b		b						
3	u								
4	l								
5	a	a			a		**a**		
6	h					h			
7	a	a			a		a		
8	m							**m**	
9	u								
10	∅								∅

fig. 8: $S_7\{a_1^4 \rightarrow a_5^6 \rightarrow m_8^7\}$

1. Strongly agglutinative languages may stack prefix and suffix morphemes around a stem such that the longest match sequence may not identify the proper-name but elements from a common template. In these circumstances, we rely on the core proper-name phonemes being matched closely and their signal will outweigh that of any morpheme structures.
2. It is possible that no one sequence will be longer than the rest i.e. there may be a number of sequences all matching the same number of characters. In these circumstances we need a way to choose between them.

Evaluating Match Sequences

Given the seven possible match sequences in our example we might ask what it is that leads us to look more favourably on a sequence that begins $\{a_1^1 \rightarrow b_2^2 \rightarrow a_5^4 \ldots\}$ than one beginning $\{a_1^1 \rightarrow h_6^5 \rightarrow a_7^6 \ldots\}$ or $\{a_1^1 \rightarrow b_2^2 \rightarrow a_7^6 \ldots\}$. We observe that our confidence in a pair of matches being sequential is based on the distance between their x and y coordinates on the matrix. Where a pair of matches are proximal (ideally adjacent) in one or both of the words our confidence rises. Conversely, there will come a point when the distance between a matched pair in one or other of the words is such that we have no confidence that the pair is part of a valid sequence. This threshold distance can become a useful processing limit. We will call it theta – θ. The value of theta is language dependent insofar as languages with longer average word lengths may require a higher value for theta but for most languages setting theta as 20 is sufficient. Let us return to our three example sequence fragments quoted above:

1. $\{a_1^1 \rightarrow b_2^2 \rightarrow a_5^4 \ldots\}$
2. $\{a_1^1 \rightarrow h_6^5 \rightarrow a_7^6 \ldots\}$
3. $\{a_1^1 \rightarrow b_2^2 \rightarrow a_7^6 \ldots\}$

Taking example 1. we construct a set of ordered pairs which represent the pairs of matched characters along the sequence: $\{(a_1^1, b_2^2), (b_2^2, a_5^4) etc \ldots\}$ The proximity of the successor to the predecessor in each pair may be expressed as the difference between the *x* and *y* coordinates for each character in the pair. Thus for (a_1^1, b_2^2) the difference between the x coordinates for each character is: $|a^x - b^x|$ (we will call this *dx*) and for the y coordinates: $|a_y - b_y|$ (which we name *dy*). For this pair $|a^1 - b^2| = 1$ and $|a_1 - b_2| = 1$ so *dx* and *dy* are both 1. For the second pair in the sequence (b_2^2, a_5^4) *dx* = 2 and *dy* = 3. We need to combine these two values such that the larger of the two has the greatest effect on our assessment of proximity. We ensure this as follows by defining d_1 as the greater of *dx* and *dy*: $d_1 = max(dx, dy)$ and d_2 as the smaller of the two values: $d_2 = min(dx, dy)$. We now combine the two values to form a single value *d* thus: $d = \theta - \left(d_1 + \left(\frac{1}{\theta} \cdot d_2\right)\right)$.

Working this for our example we find that for the first pair (a_1^1, b_2^2) d_1 and d_2 are both 1 and so we calculate *d* for this pair as: $d = 10 - \left(1 + \left(\frac{1}{10} \cdot 1\right)\right) = 8.9$. For the second pair (b_2^2, a_5^4) *dx* =

2 and $dy = 3$ so $d_1 = 3$ and $d_2 = 2$. d for this pair is therefore: $d = 10 - \left(3 + \left(\frac{1}{10} \cdot 2\right)\right) = 6.8$.

If we apply this to example 2: $\{a_1^1 \rightarrow h_6^5 \rightarrow a_7^6 \ldots\}$ we find that for the first pair $\left(a_1^1, h_6^5\right)$ d evaluates to 4.6 and for the second pair $\left(h_6^5, a_7^6\right)$ d evaluates to 8.9. Similarly for our third example $\{a_1^1 \rightarrow b_2^2 \rightarrow a_7^6 \ldots\}$ d for the first pair evaluates as 8.9 and for the second pair as 4.6.
For the value v of an entire match sequence dS we simply calculate the value for each pair of matches in the sequence and then take the product of those values: $v = \prod_{i=1}^{|dS|-1} \theta - \left(d_1 + \left(\frac{1}{\theta} \cdot d_2\right)\right)$
For the seven sequences generated by our example match this returns the following values:

$dS_1 = 46.92$
$dS_2 = 467.34$
$dS_3 = 294.77$
$dS_4 = 42664.72$
$dS_5 = 350.04$
$dS_6 = 388.13$
$dS_7 = 40.02$

It can be seen at once that we have a clear winner. When identifying a proper-name from amongst the other words around it we can count ourselves spectacularly unlucky should another word return a match sequence of similar strength to that of the name. Clearly, the shorter the name and the longer the average word length in the language the greater the chance of a spurious match but experiment suggests that for the vast majority the match-sequence generated by the proper-name will be strongest.
To express the match value for a sequence as a probability we simply calculate the value of a perfect match p for the shorter of the two words: $p = 8.9^{|dS|-1}$ and then divide v by p to render the result as a probability between 0 and 1. So for our two words *abraham* and *abulahamu* the best possible match sequence would be 6 pairs ($|abraham|$ - 1) matching every letter of *abraham* as adjacent in both items: which gives p = 496981.29. Our best sequence dS_4 gives v = 42664.72 and $P(x, y) = \frac{v}{p}$ gives a probability of 0.85 for the match being valid. The next best sequence dS_2 we find gives P = 0.009.

Appendix B: Mapping Discontinuous Structures in Natural Language – Results

Use Case 1: *Proper Name Identification*

Model Name	Finnish		Fulfulde (Niger-Congo)	Tojolabal (Mexico-Mayan)	
Abiathar	Abjatar Abjatarille	Abjatarin Abjatarilta	Abiyater	Abiatar Abiatari	
Aziel	Asielin		Ajiyel	Azieli	
Boaz	Boas Boasin Boasille		Not found	Booz Boozi	
Chedorlaomer	Kedorlaomer Kedorlaomerin Kedorlaomeria		Kedorlayomer	Kedorlaomeri Kedorlaómeri kedorlaómeri kedorla'omer	
Darius	Dareios Dareioksen Dareiokselle		Dariyus	Darío	
David	Daavid Daavidia Daavidiin Daavidkin Daavidin	Daavidille Daavidista Daavidilta Daavidilla	Daawuda	David Dabidi Davidi Davida	Dabid davidi Dvidi
Elimelech	Elimelek Elimelekin Elimelekille		Elimelek	Elimelec Elimelek Elimeleki	
Ezekiel	Hesekiel Hesekielille		Ejekiyel	Ezequiel Ezequieli	
Festus	Festus Festukselle	Festuksen	Festus	Festo	
Gehazi	Gehasi Gehasia	Gehasin Gehasille	Geehaji	Guehazi	
Haman	Haman Hamanin Hamanille	Hamanista Hamania	Haman	Amán Amani Amána	
Jezebel	Isebelin Isebel Isebelille		Ijabel	Jezabel Jezabeli Jezreel	
Lo-Ruhamah	Lo-Ruhama Lo-Ruhaman		Not found	Lo-ruhama	
Methuselah	Metuselah Metuselahin		Matusala	Matusaleni Matusalen	
Nebuchadnezzar	Nebukadnessar Nebukadnessarille Nebukadnessarin Nebukadnessaria		Buutunasar	Nabucodonosor Nabucodonosori	
Peninnah	Peninna Peninnalle	Peninnalla	Peninna	Peniná	
Rehoboam	Rehabeam Rehabeamin Rehabeamista Rehabeamille		Robo'am	Roboami Roboam	
Simon	Simon Simonin Simonia Simonille		Simon	Simon Simón Simoni	
Timothy	Timoteus Timoteus-niminen Timoteukselle	Timoteuksen Timoteusta	Timote	Timoteo	
Yahmai	Jahmai		Yakamay	Jahmai	
Zephaniah	Sefanja Sefanjalle		No text	No text	

Use Case 2: *Non-Concatenative Morphology Analysis (Classical Hebrew)*

The text of the first 11 chapters of the book Genesis [Kittel, 1997] was used as input data. Results are presented using Michigan-Claremont encoding using '_' to represent lacunae in patterns.

First Iteration Output:

Morphology:

Template	*PoS*
T.O_A_:NFH	Qal. fut. 3ppf
T.O_A_	Qal. fut. 2psm / 3psf
TO_:_W.	Qal. fut. 2ppm
TO_A_	Qal. fut. 2psm / 3psf
Y.I_:_:_W.	Qal. fut. 3ppm
Y.O_:_W.	Qal. fut. 3ppm
Y.O_A_	Qal. fut. 3ppm
YI_:_:_W.	Qal. fut. 3ppm
YO_A_	Qal. fut. 3psm
_.E_E_	n. m. coll.
_:E_O_	Qal. imp. s. m.
_:_F_IYM	n. m. p.
_:_F_	Qal. pret. 3psm
_A_:_W.	Piel fut. 3ppm
_E_E_	n. m. coll.
_F_:_FH	Qal. pret. 3psf
_F_:_W.	Qal. pret. 3pp
_F_:_	Qal. 3psm
_F_A_:T.F	Qal. pret. 2psm
_F_A_:T.IY	Qal. pret. 1ps
_F_A_	Qal. 3psm
_F_F_	Qal. pret. 3psm
_I_:_FH	Hiph. pret. 3ps
_I_:_IY	Hiph. pret. 1ps
_I_:_W.	Hiph. pret. 3pp
_O_:_FH	Qal. part. act. f.
_O_:_IYM	Qal. part. p. m. abs.
_O_A_	Qal. fut. 1ps

verification: [Davidson, 1981]

Stems:

_$_B_	שׁב	_L_Y_L_	ליל
_$_B_(_	שׁבע	_M_L_)_	מלא
_$_B_R_	שׁבר	_M_L_K_	מלך
_$_L_$_	שׁלשׁ	_M_N_X_	מנח
_$_L_X_	שׁלח	_M_R_)_	מרא
_$_M_(_	שׁמע	_N_$_)_	נשׂא
_$_M_N_	שׁמן	_N_$_M_	נשׂמ
_$_M_R_	שׁמר	_N_B_L_	נבל
_$_P_X_	שׁפח	_N_P_L_	נפל
_$_R_C_	שׁרץ	_N_S_(_	נסע
_&_M_._	שׂמ	_Q_B_R_	קבר
_&_M_L_	שׂמל	_Q_L_L_	קלל
_(_B_D_	עבד	_Q_R_B_	קרב
_(_B_R_	עבר	_R_K_$_	רכשׁ
_)_K_L_	אכל	_R_P_)_	רפא
_)_M_R_	אמר	_X_P_R_	חפר
_B_R_K_	ברך	_Y_C_)_	יצא
_C_D_Q_	צדק	_Y_K_L_	יכל
_D_B_R_	דבר	_Y_L_D_	ילד
_G_D_L_	גדל	_Y_R_)_	ירא
_H_L_K_	הלך	_Z_Q_N_	זקנ
_L_Q_X_	לקח		