# Finding regularity in Tlingit verb prefixes

Marcel K. Goh

30 April 2021

**Abstract.** [Put an abstract here. Possibly think of a better title; the one above is something I came up in five seconds that more or less describes the mumbo-jumbo my program produces. A more ambitious program would have resulted in a title of "Automatic segmentation and glossing of Tlingit verbs". But alas my program does no such thing.]

## 1. Introduction

[Probably an intro paragraph describing the Tlingit language as a whole, where it fits into language families, and the current linguistic situation. Briefly mention dialectal issues. Lure the reader in. Mention related work, etc. References I will use are [2], [3], [4], [1], and [5], and some others as well.]

## 2. Hidden and observed states

[Formalise the problem. Describe how the format of [3] allows the tables to be decomposed into a dictionary that maps sequences of underlying morphemes to sequences of actual observed prefixes. We study the inverse problem. Given a sequence of prefixes, how do we identify the original morphemes that they represent? Describe what the tags mean and how they will help us check when our rewrite approach has succeeded.]

**Strings and tags.** Formally, our program deals with the monoid of all strings over a finite alphabet $A$ (the set of all letters in the Tlingit orthography), with the binary operation of concatenation; strings in this set will be written in a fixed-width typeface to distinguish them from the ordinary text of the paper. As an exception to this convention, we use $\epsilon$ to denote the identity element (the empty string). The data found in [3] is labelled, in the sense that every string has been partitioned into substrings, each tagged with the type of prefix they represent. These tags are important to our program, so we will list all of them here.

i) A *disjunct prefix* is tagged with $Q$. This may be a qualifier prefix, an incorporated noun prefix, or an object prefix.

ii) The *irrealis prefix* is tagged with $R$. The only two possibilities here are $R(\texttt{u})$ and $R(\texttt{w})$.

iii) *Aspect prefixes* are tagged with $A$. An example is the perfective prefix, which can either be $A(\texttt{wu})$ or $A(\texttt{u})$.

iv) The *modality prefix* is tagged with $M$. The underlying form is always $M(\underline{\texttt{g}})$.

v) *Subject prefixes* are tagged with $S$.

vi) The passive, antipassive, or middle voice is indicated by a `d-` *prefix*, which is tagged with $D$.

vii) Any of the `s-`*,* `l-`*, or* `sh-` *prefixes* are classified under the $F$ tag, and we only deal with the `s` case, though it is noted in [3] that the phonological patterns are more or less analogous in the other two cases.

viii) The stative `i-`*prefix* is given the $I$ tag.

ix) *Epenthesis* is indicated by the $E$ tag.

Our program makes crucial use of these tags to detect when a sequence of observed prefixes corresponds exactly to a sequence of morphemes. We denote by $A^*$ the set of all words formed from letters in the orthography, and if the nine tags above are viewed as functions, we can let $B$ be the union of images of $A^* \setminus \{\epsilon\}$ under each of the nine functions above (an example of an element of $B$ is $S(\texttt{yi})$). Note that we never tag the empty string $\epsilon$. Next, we consider the be the set of all nonempty sequences of elements of $B$, denoted $B^+$; for instance, the sequence $Q(\texttt{ka})S(\underline{\texttt{x}})D(\texttt{d})I(\texttt{i})$ is an element of this set. $B^+$ is a semigroup under the operation of concatenation and can be made into a monoid $S$ by artificially adjoining an (untagged) element

$\epsilon$. This identity element will be important later when we start defining rewrite rules as functions from $S$ to itself.

**Hidden and observed sequences.** Although the data that the program uses is formatted as a table, it is simpler to think of it as a map (or dictionary) from a set $H \subseteq S$ of sequences of underlying morphemes to a set $K \subseteq S$ of actual observed sequences of prefixes.

### 3. Rewrite rules

[The meat and potatoes of the paper. In this section I will describe all of the rewrite rules that my program uses to match sequences of prefixes to sequences of underlying morphemes. A step-by-step sequence of rewrite rules (dozens of them) will be elaborated, but it will also be fruitful to study what they mean individually and combinatorially. For example, sometimes we will apply and then undo the same rule multiple times to test it in combination with many other rules. Discuss the exponential blowup of such a scheme (for example, if there are only 10 rewrites we can try on a particular sequence of prefixes, there are already $2^{10} = 1024$ different combinations we must try. Luckily, for any given word, there are not too many rewrites to try — at least, that I have seen so far. (My program is not yet done.)]

The following list describes a sequence of rewrite rules that the program applies, in order, to try to resolve every sequence of prefixes. For $x, y \in S$, $x \neq \epsilon$, a *rewrite rule* $x \to y$ can be viewed as a function from $S$ to itself that replaces every instance of $x$ in a word $w \in S$ with $y$.

i) $E(*) \to \epsilon$.

The first rule removes all epenthesis from all strings in the data. This is something that we have the luxury of doing because of our tagged data, and this step already resolves 45% of all the entries found in the tables.

### 4. Analysis of rewrite rules

[In this section, I want to analyse what the global sequence of rewrites actually means for individual strings. It is expected that for a given string $w$, most rewrites do not actually affect $w$ directly, though in certain cases we may have to try and undo the same rewrite multiple times to try different combinations. It might also be interesting to see distributional data related to this (but I will have to code it up first).]

### 5. Grammar induction

[May not actually include, depending on how robust the above results actually are. If I end up not doing this section, it will merge with the next section. I was thinking of including a section about the general problem of grammar induction in formal language theory and artificial intelligence. My program does not come anywhere close to proper grammar induction, but I would like to make comparisons, because the goals are largely aligned with ours. Our data just happens to be at a smaller-scale and it also comes pre-tagged. In some sense, the way our data is set up makes the problem a lot easier than the general problem of grammar induction. However, if we want to extend the program to "real-world" data such as verbs found in the text corpus, then we might have to fall back on tried-and-tested approaches found in the literature (which may stumble because of the paucity of our data).]

### 6. Further directions

[Related to the grammar induction section. Reveal all of the shortcomings of our naïve program and discuss what can be done to extend the algorithm to work on the verbs found in the corpus, not just the nicely formatted verbs found in the prefix charts.]

### References

[I have a convoluted Bash script that alphabetises, numbers, and outputs TeX for references. I know this is not the preferred linguistics style, but I will keep it this way unless you would specificially like it changed, since I don't currently have the time to modify my Bash script.]

[1] Antti Arppe, Christopher Cox, Mans Hulden, Jordan Lachler, Sjur N. Moshagen, Miikka Silfverberg, and Trond Trosterud, "Computational modeling of the verb in Dene languages. The case of Tsuut'ina," *Working papers in Athabascan Linguistics* Red Book (2017), 51–68.

[2] James A. Crippen, *Tlingit text corpus* (GitHub repository, `https://github.com/jcrippen/tlingit-corpus`, 2015).

[3] James A. Crippen, *Tlingit verb prefixes* (GitHub repository, `https://github.com/jcrippen/tlingit-verb-prefixes`, 2020).

[4] Mans Hulden and Sharon T. Bischoff, "An experiment in computational parsing of the Navajo verb," *Coyote Papers* **16** (2008), 110–118.

[5] Andrew J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory* **13** (1967), 260–269.