A Finite-State Morphological Analyzer for Central Alaskan Yup'ik

Lonny Alaskuk Strunk

A thesis submitted
in partial fulfillment to
the requirements for the degree of

Master of Science

University of Washington

2020

Reading Committee:

Emily M. Bender, Chair

Sharon Hargus

Program Authorized to Offer Degree:
Department of Linguistics

University of Washington

Abstract

A Finite-State Morphological Analyzer
for Central Alaskan Yup'ik

Lonny Alaskuk Strunk

Chair of the Supervisory Committee:
Professor Emily M. Bender
Linguistics

This thesis presents a detailed description of the design and implementation of a finite-state morphological analyzer for Central Alaskan Yup'ik (ISO 639-3: esu). Using a dictionary and a grammatical description of the language, I implemented the grammatical processes with a test-driven development approach using test-by-generation and test-by-analysis. I evaluated the morphological analyzer on two sources of texts, a collection of culture books and the Yup'ik Bible. The evaluation results show the analyzer has a parsing coverage of 89.1% of tokens in the culture books corpus and 90.4% of tokens in the Yup'ik Bible corpus. My hope is that this finite-state morphological analyzer may be integrated into other language technologies and becomes beneficial to the Central Alaskan Yup'ik community and future language learners.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CAY	Central Alaskan Yup'ik
FST	Finite-State Transducer
GCY	General Central Yup'ik
HBC	Hooper Bay-Chevak dialect
HFST	Helsinki Finite-State Transducer toolkit
IYU	Inuit-Yupik-Unangan
TDD	Test-Driven Development
IGT	**Interlinear Glossed Text**
>	Direction of transitivity or possession
1	First Person
2	Second Person
3	Third Person
4	Fourth Person ((3R) Reflexive-Third Person)
ABM	Ablative-Modalis Case
ABS	Absolutive Case
CNN	Connective Mood
CNNbc	Consequential/Causal Connective Mood: 'because'
CNNbf	Precessive Connective Mood: 'before'
CNNif	Conditional Connective Mood: 'if, when (in the future)'
CNNqs	Quasi-connective Mood
CNNth	Concessive Connective Mood: 'although, even though, even if'

CNNst	Stative Connective Mood
CNNwl	Second Contemporative/Durative Connective Mood: ‘while’
CNNwn	First Contemporative Connective Mood: ‘when (in the past)’
CNNwv	Contingent/Constantive Connective Mood: ‘whenever’
DES	Desiderative
DU	Dual Number
EQL	Equalis Case
EXCLAM	Exclamative
FUT	Future Tense
IND	Indicative Mood
INT	Interrogative Mood
LOC	Localis Case (Locative Case)
OPT	Optative Mood
NEG	Negative
PL	Plural Number
POSS	Possessive Marker
PST	Past Tense
PTP	Participial Mood
REL	Relative Case
SG	Singular Number
SUB	Subordinative Mood (Appositional Mood)
TER	Terminalis Case (Allative Case)
VIA	Vialis Case (Perlative Case)

ACKNOWLEDGMENTS

First, I would like to thank the Yup'ik elders and researchers who worked on documenting Central Alaskan Yup'ik. Without Jacobson's work on the dictionary and grammar book, I could not have done this project.

I would also like to thank my advisor Emily M. Bender, whose support and guidance with this project have been invaluable. She has encouraged me and held me accountable with working on my project and writing this thesis and I am forever grateful.

I want to thank Sharon Hargus for agreeing on being my second reader and her valuable feedback on my thesis. I also want to thank Lane Schwartz for inviting me to travel with him and work on indigenous languages, and Francis Tyers for introducing me to *twolc* and setting up the initial FST stack. I would also like to thank my CLMS cohort especially Seraphina Goldfarb-Tarrant and Marcus Martinez for being my friends during times of great stress.

I am grateful for the Yup'ik courses and my classmates at UAF for allowing me to continue learning my heritage langauge. I am especially grateful because they introduced me to my best friend, *Akiuk* (Alexander King), who has helped me to grow and learn and have opinons. I rely on his wisdom and guidance and constant friendship both before and during this project.

Finally, I want to thank my parents and family for their constant support of my goals to continue my education. Thank you for instilling in me a love for language and learning.

Chapter 1

INTRODUCTION

Large written and spoken corpora are used to create various language technologies including electronic dictionaries, spell-checkers, grammar-checkers, speech synthesizers, and speech recognition for majority languages. Endangered or indigenous languages on the other hand, which can be morphologically complex languages, have little or no written and spoken corpora which are necessary for techniques that majority language technologies use. However, many indigenous and endangered languages do have basic lexical and grammatical descriptions which can be used to create finite-state morphological analyzers, a mature technology integrated into functional practical applications (Arppe et al., 2016).

Finite-state morphological analyzers are a rule-based technology that computationally models the morphology of a natural language. This technology can model many morphologically complex phenomena including morphotactics and morphophonological processes. A finite-state morphological implementation can be used for analysis, taking a fully inflected word as input and giving a morpheme separated underlying representation as output, or used for generation, taking the underlying form as input and giving a fully inflected word as output (Beesley and Karttunen, 2003a). Integrating this technology into an electronic dictionary, for instance, will allow the user to look up morphologically complex words and allows the dictionary to generate full paradigms of inflected words.

For this thesis, I created a finite-state morphological analyzer for my heritage language, Central Alaskan Yup'ik. Central Alaskan Yup'ik is a polysynthetic indigenous language with a complex morphology, the primary process being recursive suffixation. My hope is that this project will lead to further useful applications such as an intelligent dictionary and spell-checker, and that this project may inspire speakers of indigenous languages to create

their own finite-state morphological analyzers to improve the digital presence of indigenous languages on the internet.

This thesis begins with a language and literature survey, first with a description of Central Alaskan Yup'ik and the morphological phenomena in Chapter 2, and then with a description of finite-state transducers and the tools needed to create the morphological analyzer in Chapter 3. In Chapter 4, I describe the design of the finite-state morphological analyzer which includes the lexicon, the underlying form representation, and the individual components of the project. In Chapter 5, I describe the two evaluation methods, the test suite used in the development phase and a corpus test used to evaluate the quality of the analyzer. Finally, in Chapter 6, I report on the results of the test suite and corpus test as well as an error analysis of the corpus test results.

Chapter 2

CENTRAL ALASKAN YUP'IK AND MORPHOLOGY

Central Alaskan Yup'ik /jupːik/ (CAY) language (ISO 639-3: esu) (*Yugtun* / *Yugcetun*) is a member of the Inuit-Yupik-Unangan (IYU) (historically known as Eskimo-Aleut[1]) language family. It is spoken in the Yukon-Kuskokwim Delta and Bristol Bay of Southwest Alaska. CAY includes a few mutually intelligible dialects. The majority of speakers use the General Central Yup'ik (GCY) dialect which encompasses the majority of the CAY area. GCY can be further subdivided into areas with regional lexical differences. The other dialects of CAY include the Hooper Bay-Chevak dialect, Nunivak Island dialect, Norton Sound dialect, and Egegik dialect. These dialects include derivational and inflectional suffixes that have variations from GCY in their morphophonology, prosody, or form (Jacobson, 2012).

Suffixation is the main process used in the morphology of Central Alaskan Yup'ik. Woodbury (2014) describes the (virtually) exclusive technique of suffixation as the 'morphological orthodoxy' of Inuit-Yupik languages in his paper by the same name:

> Yupik-Inuit languages have one pervasive morphological process, recursive suffixation to a base, and—normally—a corollary scope rule according to which any suffix is an operator or modifier with scope over exactly the base to which it was added. This pattern is both prolific and exclusive: there is (almost) no prefixation, no mutation, ablaut, reduplication, nor any base-base or (practically any) word-word compounding. Moreover the pattern has apparently been historically persistent, since it dominates all known members of Yupik-Inuit and more distantly-related Aleut as well. (pg. 151)

[1]Both the terms Eskimo and Aleut are considered derogatory by some members of the communities of which they refer to. For the purposes of this thesis, the terms will only be used in the titles of cited works.

This highly productive process of recursive suffixation results in, morphologically, an infinite number of possible words (Miyaoka, 2012), but according to Jacobson (1984), CAY words may have up to six derivational suffixes but rarely more. CAY suffixation is semantically conditioned by the corollary scope rule and it is semantic compatibility that is the guiding structural principle (Miyaoka, 2012).

2.1 Central Alaskan Yup'ik Resources

My project utilizes the grammatical descriptions that were developed in the 1960s and 1970s at the University of Alaska's Alaska Native Language Center in Fairbanks and the Yup'ik Language Workshop in Bethel. The main framework of the modern grammar and orthography was developed by many linguists and native Yup'ik speakers including Michael Krauss, Irene Reed, Martha Teeluk, Osahito Miyaoka, Paschal Afcan, and Stephen Jacobson. Their work resulted in the first modern grammar and orthography books, *Yup'ik Eskimo Grammar* (Reed et al., 1977) and *Yup'ik Eskimo Orthography* (Miyaoka and Mather, 1979). These textbooks were used primarily in the university classroom to teach basic vocabulary, literacy, and grammatical functions. Later, Woodbury (1981) published work on the *Study of the Chevak Dialect of the Central Yup'ik Eskimo* using the grammatical and orthographic conventions established by Reed et al. (1977). Another work that was developed during this period was *Qaneryaurci Yup'igtun* by Hensel et al. (1985), a grammar textbook focused on practical communication.

The next major publication came from the first edition of the *Yup'ik Eskimo Dictionary* (Jacobson, 1984) which featured a compiled list of CAY stems, derivational suffixes, and enclitics with English definitions, inflectional suffix tables, and English-Yup'ik lookup sections using the modern grammar and orthographic conventions. A decade later, Jacobson (1995) published *A Practical Grammar of the Central Alaskan Yup'ik Eskimo Language* which is an updated and expanded grammar book of Reed et al. (1977), again to offer an accessible grammar textbook for language classrooms.

In 2012, two major works on CAY were published. The first work was the second edition

of the *Yup'ik Eskimo Dictionary* (Jacobson, 2012). This features many more lexical entries, updated definitions and analyses, protoforms, dialectal information, and many more sentence examples. The second work was *A Grammar of Central Alaskan Yupik (CAY)* by (Miyaoka, 2012). It is a massive comprehensive descriptive grammar resource of CAY.

In addition to the grammatical works previously listed, there are a number of other publications written in CAY. Many of these are bilingual volumes of traditional stories and narratives compiled from interviews with Yup'ik elders from the Yukon-Kuskokwim Delta such as Paul John and Frank Andrew. They were edited by anthropologist Ann Fienup-Riordan and transcribed and translated by native speakers, Alice Rearden and Marie Meade (John et al., 2003; Fienup-Riordan and Rearden, 2011, 2013, 2014, 2016a,b; Fienup-Riordan et al., 2017, 2018). Another bilingual book of compiled stories was published by the Lower Kuskokwim School District (Tennant and Bitar, 2000). Recently one of the major publications is the completed translation of the Bible into CAY, called *Tanqilriit Igat* (American Bible Society, 2014).

In this project, I use Jacobson's (1995) grammatical description and Jacobson's (2012) second edition of the dictionary to develop the morphological analyzer. I also used Miyaoka's (2012) descriptive grammar when I needed a deeper understanding into specific grammatical phenomena. In addition, I created a corpus of words taken from the bilingual publications and Yup'ik Bible to evaluate the morphological analyzer.

2.2 Orthography

My morphological analyzer uses the current writing system of CAY that was developed in 1960s at the University of Alaska Fairbanks by many linguists and native Yup'ik speakers including Michael Krauss, Irene Reed, Martha Teeluk, Osahito Miyaoka, Paschal Afcan, and Stephen Jacobson (Miyaoka and Mather, 1979). This orthography was developed to have a one-to-one correlation between the written word and the pronunciation, where "each written word can be pronounced in only one way and most spoken words can be written in only one way" (Reed et al., 1977). This orthography uses a subset of the English alphabet using 18

letters and a few diacritics and punctuation marks to represent all of the graphemes in CAY. The graphemes are arranged in Tables 2.2 and 2.1 where the grapheme is paired with the IPA symbol in slashes "/".

Many of the consonants are represented as two letters for a single sound, either a doubled letter such as the voiceless fricatives or as a combinations of letters such as the velar nasal and labialized velar and uvular fricatives. The diacritics in CAY include the ligature tie for labialized velar and uvular fricatives and the macron or bar above voiceless nasals.

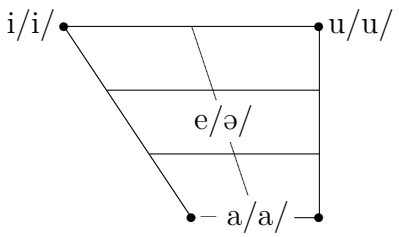

Table 2.1: Vowel Chart

There are four vowels in CAY. The vowels *a*, *i*, and *u* are called full vowels or prime vowels. The other vowel *e* is a reduced vowel because it is never doubled, next to another vowel, or at the end of a word. The full vowels may occur doubled or any two different full vowels may occur next to each other in a diphthong.

The apostrophe (') and hyphen (-) are also used in the orthography. The many uses of the apostrophe will be explained in the orthographic rules, Section 2.2.1, and the hyphen is used to separate enclitics from the rest of the word or as a separator for code-mixing English words into Yup'ik words (Miyaoka and Mather, 1979).

2.2.1 Orthographic Rules

There are a few orthographic rules in CAY where letters are pronounced differently from the graphemes given in Tables 2.2 and 2.1 due to their position in the word. These orthographic rules take advantage of the regularities in the phonology and prosody of the language and

	bilabial	labio-dental	alveolar	post-alveolar	palatal	velar	labio-velar	uvular	labio-uvular
stop	p/p/		t/t/			k/k/		q/q/	
nasal	m̄/m̥/ m/m/		n̄/n̥/ n/n/			n̄g/ŋ̊/ ng/ŋ/			
affricate			[ts]	c/ʧ/					
fricative		vv/f/ v/v/	ss/s/ s/z/			gg/x/ g/ɣ/	w/x^{w}/ u͡g/ɣ^{w}/	rr/χ/ r/ʁ/	u͡rr/χ^{w}/ u͡r/ʁ^{w}/
lat. fric.			ll/ɬ/ l/ɮ/						
approx.					y/j/		[w]		

Table 2.2: Consonant Chart

integrate the sound changes in the orthography (Jacobson, 1995). Many people have reported that, "they have found it unexpectedly easy to learn to read and write in Yup'ik even without getting any formal instruction" (Reed et al., 1977). All orthographic rules below are taken from Jacobson's (1995) grammar book.

1. Phonologically conditioned devoicing on consonants (also known as automatic devoicing) — Since voicelessness is predictable in certain contexts, the orthography doesn't represent it, rather, the symbol for the voiced consonant is used in the following cases: (a) A fricative is written single next to a stop and pronounced voiceless (1a,b). (b) A fricative that follows a voiceless fricative is written single but pronounced voiceless (1c). (c) A nasal that follows a stop or voiceless fricative is written without an accent mark but pronounced voiceless (1d,e). (d) An *r* at the end of a word is written single but pronounced voiceless (1f). (e) An *s* at the beginning of a word is written single but pronounced voiceless (1g).

(1)

	Orthographic	*Graphemic*[2]	*Phonemic*
a.	puqla	puqlla	puqɬa
b.	ayagtuq	ayaggtuq	ajaxtuq
c.	ayallruuq	ayallrruuq	ajaɬχuːq
d.	cavutnguuq	cavutn̄guuq	ʧavutŋ̊uːq
e.	allngiq	allm̄giq	aɬŋ̊iq
f.	angyacuar	angyaccuarr	aŋ̊jaʧːuaχ
g.	suugiuq	ssuugiuq	suːɣiuq

2. Rhythmic Lengthening — In a sequence of simple open syllables ([#|C]V), every second syllable gets rhythmically lengthened unless it is at the end of the word.

[2]The orthographic form is the surface form that corresponds to the modern orthography. The graphemic form represents what the form would look like before all of the orthographic rules in this section are applied. The phonemic form is the IPA representation of the surface form.

(2)

	Orthograpic	*Graphemic*	*Phonemic*
a.	nalluyagucaqunaku	nalluuyaguucaquunaku	naɬuːjaɣuːʧaquːnaku
b.	angyaliqataraqama	angyaliiqataaraqaama	aŋjaliːqataːʁaqaːma
c.	qayaliciqngatuten	qayaaliciqngatuuten	qajaːliʧiqŋatuːtən

3. Phonologically conditioned gemination on consonants (also known as automatic gemination) — a consonant is geminated in the following contexts: (a) A consonant preceded by a single vowel which is not rhythmically lengthened, and is followed by two vowels (alike or unlike), is geminated (3a,b). (b) If a rhythmically lengthened *e* separates two same or almost same consonants, then it is kept and the consonant that follows it is geminated (3c).

(3)

	Orthographic	*Graphemic*	*Phonemic*
a.	aciani	acciani	aʧːiani
b.	nuniini	nunniini	nunːiːni
c.	tumemi	tumemmi	tuməmːi

4. Apostrophe — the meaning of an apostrophe depends on the location in the word. The uses are as follows: (a) An apostrophe between two consonants is either used to separate *n* and *g* to avoid misreading (4a), or to separate a voiced fricative or nasal from a stop or voiceless consonant (4b). (b) An apostrophe between a consonant and the following vowel indicates gemination (4c). (c) An apostrophe between a vowel and the following consonant indicates stress on the syllable not predictable from rhythmic stress, "ghost stress" (see subsection *Prosody and Rhythmic Stress* below) (4d). (d) An apostrophe between two vowels indicates the lack of gemination on the preceding consonant (4e). (e) An apostrophe at the end of a word is either used to indicate a segment has been deleted (4f), or when a stop is replaced with a voiced fricative (4g).

(4)

	Orthograpic	*Graphemic*	*Phonemic*
a.	tan'gurraq	tan.gurraq	tanɣuχaq
b.	tep'lek	teplek	təplək
c.	ner'uq	ner.ruq	nəʁːuq
d.	qavartu'rtuq	qavarturtuq	qawaχtuχtuq
e.	apa'urluq	apaurluq	apauʁluq
f.	qaill'	qaill	qaiɬ
g.	arnar'	arnar	aʁnaʁ

Since the morphological analyzer uses the orthographic form as input, the orthography rules will have to be taken into account when implementing the other morphophonological phenomena.

2.2.2 Prosody and Rhythmic Stress

In addition to prosodic phenomena such as rhythmic lengthening and automatic gemination, which are not indicated in the writing system because they are predictable, there is another prosodic phenomenon called rhythmic stress. Stress in CAY is pronounced with greater volume and emphasis than the other syllables. Stress is only indicated in the orthography when it is not predictable from the rhythmic stress rules, in which case the stressed syllable is marked with an apostrophe. There are two different ways to analyze rules for rhythmic stress, one analyzes rhythmic lengthening and automatic gemination as specific consequences of stress (Jacobson, 1995, p.14), and the other considers rhythmic stress as an extension of the system for rhythmic lengthening (Jacobson, 1995, p.55). I will explain the latter system of rules for rhythmic stress, which is rephrased from Jacobson's (1995, p.55) grammar book, in (5).

(5) Rules for rhythmic stress

1. A non-final syllable will receive stress if: (note that more than one may apply)
   (a) it contains a rhythmically lengthened vowel (and kept hatted *e*), or
   (b) it contains two vowels, or

(c) it is a syllable preceding syllables with two vowels, or
(d) it is a closed word-initial syllable

2. In a sequence of two or more syllables not yet stressed by Rule 1, every second syllable will receive stress if that syllable is closed, not assigning stress to the word final syllable.

3. If stress would be assigned to an open syllable following a closed syllable from Rule 2, then stress is reassigned to that preceding closed syllable and counting for Rule 2 begins again at that open syllable.

To demonstrate these rules (examples taken from Jacobson (1995)), given the word *iqvallrunrituq* 's/he wasn't picking berries', separated into syllables as *iq.vall.run.ri.tuq*, rules 1a-1c do not apply but 1d applies and assigns stress to *iq*. The sequence of syllables for rule 2 begins at *vall* and assigns stress to *run*, then skips *ri* and does not assign stress to *tuq* because it is the word-final syllable. Altogether the assigned stress indicated with an accent marker on the vowel is *íq.vall.rún.ri.tuq*.

Another example is the word *qa.yar.pa.li.yull.ruu.nga* 'I wanted to make a big kayak'. The syllable *li* is stressed from 1a because it is rhythmically lengthened, *ruu* is stressed from 1b, and *yull* is stressed from 1c to give *qa.yar.pa.lî.yúll.ruú.nga*. From rule 2, the sequence *qa.yar.pa* is considered and *yar* is assigned stress. Altogether the assigned stress is *qa.yár.pa.lî.yúll.ruú.nga*.

A final example is the word *i.kam.rar.pa.lin.ri.tuq* 's/he isn't making a big sled'. Rule 1 does not assign any stress and so rule 2 applies to the whole word. It assigns stress to the second syllable *kam* and the next second syllable to be considered is the open syllable *pa*. Thus rule 3 applies and assigns stress to the preceding syllable *rar* and the count for rule 2 begins again at *pa*. Thus the next second syllable is *lin* and does not assign stress to *tuq* since it is word-final. Altogether the assigned stress is *i.kám.rár.pa.lín.ri.tuq*.

2.3 Morphotactics

A major component of the morphological analyzer is determining what combinations of stem and suffixes create valid words in CAY. This includes the inventory of noun and verb

inflectional suffixes, the categories of derivational suffixes, and the valency types for verb stems and derivational suffixes.

CAY words are categorized into three main types: nouns, verbs, and particles. Nouns and verbs have an obligatory inflectional suffix, while particles, the catch-all term, are (primarily) without any derivational or inflectional suffixes. Nouns and verbs have similar word morphology as shown in (6).

(6) STEM + DERIVATIONAL SUFFIX + INFLECTIONAL SUFFIX + ENCLITIC

CAY is, with a couple exceptions, a suffix-only language where a word begins with a single stem of either a noun or a verb. The stem is then optionally followed by a series of derivational suffixes and ends with an obligatory inflectional suffix. There may optionally be additional enclitics at the end of the word that are marked with a hyphen in the orthography and do not affect the morphophonology. The simplest noun or verb will consist of a stem and an inflectional suffix (Miyaoka, 2012).

(7) a. *angya-rpi-it=llu*
boat-big-ABS.3PL=and
'and the big boats'

b. *cali-llru-uq=gguq*
work-PST-IND.3SG=reported
'It is said that s/he worked.'

Inflectional suffixes can be sometimes analyzed as having up to three parts: the case/mood, the first argument person and number, and the second argument person and number. But many inflectional suffixes are portmanteaus, i.e. they express case/mood, person and number but cannot be analyzed further into smaller segments expressing each. Therefore, the convention is to treat the inflectional suffix as one (Mithun, 2012).

2.3.1 Noun Inflection

CAY nouns are inflected for case and number (SG, DU, PL), and additionally marked for person (1, 2, 3, 4 (i.e. reflexive 3rd person)) and number of the possessor. CAY nouns can take two core cases and five oblique cases (Jacobson, 1995; Miyaoka, 2012).

(8) **Noun Cases**

Core Cases

1. Absolutive (ABS)
2. Relative (REL)[3]

Oblique Cases

3. Ablative-Modalis (ABM) - 'from' and indefinite object
4. Terminalis (TER) (aka Allative) - 'to'
5. Localis (LOC) (aka Locative) - 'at/in'
6. Vialis (VIA) (aka Perlative)- 'through/with'
7. Equalis (EQU) - 'as/like'

Below is a sample of the types of inflections on nouns.

(9)			
	angyaq	boat\ABS.SG	'a boat'
	angya-ma	boat-REL.1SG>SG	'my boat'
	angya-ni	boat-LOC.PL	'at the boats'
	angya-vnun	boat-TER.2SG>SG	'to your boat'

2.3.2 Verb Inflection

CAY verbs are inflected for person (1, 2, 3, and/or 4 (i.e. reflexive 3rd person) where applicable) and number (SG, DU, PL) for subject in intransitive verbs, and both subject and object in transitive verbs. CAY verb moods consist of four independent moods and two dependent moods (Jacobson, 1995; Miyaoka, 2012).

---

[3]The relative case may also be called the ergative-genitive case.

(10) **Verb Moods**

Independent Moods

1. Indicative (IND) - statements
2. Interrogative (INT) - questions
3. Optative (OPT) - commands
4. Participial (PTP) - special functions

Dependent Moods

5. Subordinative (SUB) (aka Appositional) - cosubordinate clause
6. Connective (CNN) - 'because', 'whenever', 'when', 'even if', ...

Below is a sample of the types of inflections on verbs.

(11)			
	ner'-uq	eat-IND.3SG	'he/she/it is eating'
	ner-ai	eat-IND.3SG>3PL	'he/she/it is eating them'
	ner-i	eat-OPT.2SG	'eat!'
	ner-luni	eat-SUB.4SG	'he/she/it is eating ...' (cosubordinate clause)
	ner-ngavgu	eat-CNNbc.2SG>SG	'because you are eating it'

2.3.3 Derivational Suffixes

Derivational suffixes are separated into four major categories: verb-elaborating ([V→V]), noun-elaborating ([N→N]), nominalizing ([V→N]), and verbalizing ([N→V]). There are a large but finite number of derivational suffixes but their recursive nature allows for very long complex words (Miyaoka, 2012).

(12)					
a.	[V→V]	verb-elaborating	*-llru-*	*nere-llru-uq*
			-PST-	eat-PST-IND.3SG
				's/he ate'
b.	[N→N]	noun-elaborating	*-rpag-*	*angya-rpak*
			-big-	boat-big\ABS.3SG
				'big boat'
c.	[V→N]	nominalizing	*+(s)te-*	*nere-sta*
			-one.who-	eat-one.who\ABS.3SG
				'one who eats'; lexicalized: 'louse'

d.	[N→V]	verbalizing	*:~(ng)u-*	*angya-u-guq*
			-be-	boat-be-IND.3SG
				'it is a boat'

The limiting factor to the recursive power of the suffixes is that suffixes obey the semantic restrictions of the corollary scope rule. The corollary scope rule is defined by Woodbury (2014) as, "A suffix is an operator or modifier with scope over exactly the base to which it was added." Because suffixation is recursive, each new suffix has semantic scope over the word up to the base to which it was added. They may cycle between becoming noun and verb bases such as Example (13).

(13) a. *ner'*]$_V$*-uq*
eat-IND.3SG
's/he eats/is eating'

b. *nere*]$_V$-***sta***]$_N$
eat-**one.who**\ABS.SG
'**one who** eats'; lexicalized: 'louse'

c. *nere*]$_V$*-ste*]$_N$-***rpak***]$_N$
eat-one.who-**big**\ABS.SG
'**big** louse'

d. *nere*]$_V$*-ste*]$_N$*-rpa*]$_N$-***u***]$_V$*-guq*
eat-one.who-big-**be**-IND.3SG
'**it is** a big louse'

e. *nere*]$_V$*-ste*]$_N$*-rpa*]$_N$*-u*]$_V$-***llru***]$_V$*-uq*
eat-one.who-big-be-**pst**-IND.3SG
'it **was** a big louse'

2.3.4 Verb Stem Valency

In CAY, there are various valency types for verb stems and many valency changing derivational suffixes that influence the morphotactics and morphosyntax/semantics of CAY verbs. There are two different grammatical analyses for this grammatical feature, one analysis from Miyaoka 2012 and the other analysis from Mithun 2000 and Jacobson 2012.

In Miyaoka's (2012) analysis of verb stem valency, he focuses on the semantic arguments of the verb and not the arguments expressed in the inflection. The inflection of the verb has either one argument indexed (intransitive) or two arguments indexed (monotransitive) even though the verb may have three (ditransitive) or more arguments (multi-valent) from valency-increasing suffixes. This means that a verb has either an intransitive inflection with just the subject expressed (14a) or a transitive inflection with both the subject and object expressed (14b), but there may be more arguments expressed with other noun cases as shown in Example (14c).

(14) a. *qimugta* *tai-guq*
dog\ABS.SG come-IND.3SG
'the dog came' / 'the dog is coming'

b. *qimugte-m* *neqerrluk* *ner-aa*
dog-REL.SG dryfish\ABS.SG eat-IND.3SG>3SG
'the dog ate the dryfish' / 'the dog is eating the dryfish'

c. *angute-m* *cikir-aa* *qimugta* *neqerrlug-mek*
man-REL.SG give-IND.3SG>3SG dog\ABS.SG dryfish-ABM.SG
'the man gave the dog dryfish' / 'the man is giving the dog dryfish'

Verb stems, as seen in (14), come in three main types: intransitive, monotransitive, and ditransitive. There are also other subtypes for monotransitive and ditransitive as shown in (15). The [S=A,P,R,T] indicated in the subtypes represent which semantic role (agent, patient, recipient, or theme) from the transitive inflection becomes the subject that is expressed in the intransitive inflection (Miyaoka, 2012).

(15) **Verb Stem Valency (Miyaoka)**

1. Intransitive — monovalent stems — with S (subject)
2. Monotransitive — bivalent stems — with A (agent) and P (patient)
   a. Agentive type — [S=A]
   b. Patientive type — [S=P]
   c. Impersonal Patientive type — [S=P, with A_{IMP} 'it']
3. Ditransitive — trivalent stems — with A (agent), T (theme), and R (recipient)
   a. Secundative type — [S=R]

b. Indirective type — [S=T]

The monotransitive subtypes include agentive types where the agent of the transitive as shown in (14b), repeated here in (16a), becomes the subject of the intransitive as shown in (16b) when a zero-derived detransivizer makes it antipassive. The object is then demoted from a core case to an oblique case and becomes optionally expressed as a free NP. The optional object takes the ablative-modalis case and the definiteness of the object changes as well.

(16) a. *qimugte-m* *neqerrluk* *ner-aa*
dog-REL.SG dryfish\ABS.SG eat-IND.3SG>3SG
'the dog ate the dryfish' / 'the dog is eating the dryfish'

b. *qimugta* *ner'-uq* *neqerrlug-mek*
dog\ABS.SG eat-IND.3SG dryfish-ABM.SG
'the dog ate dryfish' / 'the dog is eating dryfish'

The next two monotransitive subtypes are related because they are both patientive types. The patientive types are defined as having the patient of the transitive as shown in (17a) becoming the subject of the intransitive as shown in (17b) when a zero-derived detransivizer makes it passive. This passivization process drops the agent and can be expressed, although rarely, as a free NP in the terminalis case (Miyaoka, 2012).

(17) a. *mikelngu-um* *saskaq* *kuv-aa*
child-REL.SG cup\ABS.SG spill-IND.3SG>3SG
'the child spilled the cup'

b. *saskaq* *kuv'-uq*
cup\ABS.SG spill-IND.3SG
'the cup spilled'

c. *nanvaq* *ciku-a*
lake\ABS.SG freeze-IND.3SG>3SG
'it [A_{IMP}] froze the lake, i.e. the lake is (now) frozen'

d. *nanvaq* *ciku-uq*
lake\ABS.SG freeze-IND.3SG
'the lake is (still) freezing'

The impersonal patientive type, which can be considered a subtype of the patientive type, shown in (17c) and (17d), is inflected with an impersonal agent in the transitive and then loses the impersonal agent in the intransitive inflection. The impersonal agent is only expressed in the inflection and not also as a separate noun in the relative case. This impersonal agent is like an uncontrollable natural and supernatural force or process used with words like 'to freeze, to dawn, to rot' (Miyaoka, 2012).

Ditransitive verb stems have two different subtypes. The first subtype is the secondative type, shown in (14c), where the recipient is the object in absolutive case and the theme is in ablative-modalis case in the transitive inflection. The second subtype is the indirective type, shown in (18b), where the theme is the object in absolutive case and the recipient is in terminalis case in the transitive inflection. When these verb subtypes take the intransitive inflection, they either act like they have undergone passivization in the patientive type, shown in (18a) and (18c), where the agent is dropped and their object in the absolutive case becomes the subject, or they act like they have undergone reflexivization in the agentive type, shown in (18d), where the agent becomes the subject in absolutive case and the object references the agent in the ablative-modalis case (Miyaoka, 2012).

(18) a. *ak'a* *ciki-llru-uq* *angun* *neq-mek*
already give-PST-IND.3SG man\ABS.SG food-ABM.SG
'the man was already given food'

b. *tun-aa* *arna-m* *akutaq* *angut-nun*
give-IND.3SG>3SG woman-REL.SG ice.cream\ABS.SG man-TER.PL
'the woman gave/sold the ice cream to the men'

c. *ak'a* *tune-llru-uq* *akutaq* *angut-nun*
already give-PST-IND.3SG ice.cream\ABS.SG man-TER.PL
'the ice cream was already given/sold to the men'

d. *tun-'uq* *ellmi-nek*
give-IND.3SG himself/herself-ABM.3RSG
'he gave himself (e.g. to God)'

Miyaoka (2012) notes that the three-way classification of stems is not always clear cut. He states, "monovalents can occur with transitive inflection to a considerable extent but with much idiolectal (rather than local) variation, while there are chiefly bivalent stems that are rarely used with transitive inflection by many speakers but only intransitively instead (again with much variation)" (Miyaoka, 2012, p.148).

In Mithun's (2000) and Jacobson's (2012) analyses of verb stem valency, they focus on the arguments expressed in the inflection and not the semantic arguments of the verb. There are intransitive-only, transitive-only, and ambitransitive inflection as shown in (19), with subtypes for ambitransitive with agentive and patientive types with an elemental subtype for patientive.

(19) **Verb Stem Valency (Jacobson & Mithun)**

1. Intransitive-only verbs — with S (subject)
2. Transitive-only verbs — with A (agent) and P (patient)
3. Ambitransitive verbs — with A (agent) and P (patient)
   a. Agentive verbs — [S=A]
   b. Patientive verbs — [S=P]
   c. Elemental verbs — [S=P, with A_{IMP} 'it']

The intransitive-only verb category is the same as the monovalent stems in Miyaoka's analysis and the agentive and patientive verb categories are the same as well. The elemental verb category is the same as Miyaoka's impersonal patientive type. The categories that differ include the transitive-only verb category for Jacobson and Mithun's analysis and the ditransitive categories for Miyaoka. Miyaoka's ditransitive stems are classifed by Jacobson and Mithun as either ambitransitive or transitive-only verbs and some monotransitive stems are classified as transitive-only verbs. For example, the verb stem *cikir-* 'to give' is considered transitive-only and *tune-* 'to give' is considered patientive in the dictionary (Jacobson, 2012).

The transitive-only verb category in Jacobson and Mithun's analysis does not correspond to anything in Miyaoka's classification. The transitive-only verb category includes verbs that "do not normally take intransitive endings directly. ... Some of the verbs in this group can take intransitive endings, but only marginally and in conjunction with a word such as *ellminek* 'himself' or *ak'a* 'already'. In such cases the meaning is reflexive: *ellminek assikuq* 'he likes himself'; or passive: *ak'a teguuq* 'it has already been taken'" (Jacobson, 2012, p.26).

2.3.5 Valency Changing Suffixes

Verb stems may be modified by suffixes that affect the verb valency. These valency changing suffixes may be valency-increasing, valency-decreasing, or valency-rearranging. Table 2.3 and 2.4, copied from Miyaoka (2012, p.830,832), show the different types of valency-increasing and valency-decreasing suffixes. A full analysis for each of the derivational suffixes will not be given in this paper, only a sample from each of the major types of suffixes.

The valency-increasing suffixes in Table 2.3 are separated into two different categories: the simplex suffixes yield valency-modified simplex verb bases and the complex suffixes yield multi-layered complex verbs embedding the lower-clause simplex verb. The example set given in (20) uses the intransitive verb stem *tuqu-* meaning 'to die'. Example (20b) adds the simplex causative suffix to add an agent to the verb stem to create a monotransitive verb meaning 'to kill'. Then in Example (20c), the complex causative suffix is added to embed the monotransitive verb to create a complex verb meaning 'to let (someone) kill it'.

(20) a. *ungungssiq* *tuqu-uq*
animal\ABS.SG die-IND.3SG
'the animal died'

b. *angute-m* *ungungssiq* *tuqu-t-aa*
man-REL.SG animal\ABS.SG die-A-IND.3SG>3SG
'the man killed the animal'

c. *arna-m* *angut-mun* *ungungssiq* *tuqu-te-vkar-aa*
woman-REL.SG man-TER.SG animal\ABS.SG die-A-A'.make-IND.3SG>3SG
'the woman let the man kill the animal'

simplex	A (causative)	+te- @[4]+car- @∼+caar(ar)- @∼+cir- –rqe-	* occur only after monovalents or roots * often lexicalized
	E (applicative) (means / T) (place / R)	@:(u)te-	* wide range of roles, e.g. benefactive
		@:(u)teke-	* rearranging as well as increasing
		@∼+vike-	
	E (adversative)	+'(g)i-	* malefactive 'to the disadvantage of'
	A_{IMP} (necessitative)	@∼+narqe- @∼+nari-	* may also be modality marker
complex	A' (causative)	@+cete-/.vkar- @∼+cetaar-	'to make, let s.o. –
	A' (directive)	–sqe- –squma-	'to ask/tell s.o. to – '
	A' (speculative)	@∼+yuke- @∼+nayuke-	'to think that s.o. – '
	A' (reportative)	@∼+ni-	'to say that s.o. –'
	A' (ignorative)	@:(u)ciite- @:(u)ciirute-	'not to know, be unsure that/ whether s.o. – (now, no longer)'
	A' (expectant)	@–nercir-	'to wait until s.o. –'

Table 2.3: Valency Increasing Suffixes (Miyaoka, 2012, p.830)

The valency-decreasing suffixes that reduce the number of arguments by one occur in two kinds. Miyaoka (2012, p.151) states, "CAY, however, has no solid suffix whose function is passivization like the 'unreservedly productive' *-niqar-* in West Greenlandic (Fortescue 1984: 266)." The two kinds of valency decreasing suffixes shown in Table 2.4 are antipassive suffixes for patientive verbs and pseudo-passive suffixes that are restricted to certain verb bases. Non-patientive verb stems are antipassivized by zero-derivation as shown in the previous section

[4]These symbols will be explained in Section 2.4.2.

in Example (16b).

decreasing	antipassive (for patientive verbs)	+'(g)i-	* vs. applicative E; most productive
		@:(u)te-	* cf. adversative E
		@∼–kenge-	
	pseudo-passive	+(s)ciur- +(s)cir-	* dynamic; generally adversative
		+'(g)ar:∼(ng)u- +'(g)ar–ke	* stative

Table 2.4: Valency Decreasing Suffixes (Miyaoka, 2012, p.832)

The *+'(g)i-* antipassive suffix for patientive verbs is the most productive valency-decreasing suffix. The examples in (21) use this suffix to reduce the syntactic valency of the verb and take the intransitive inflection. The uninflected object can still be indicated by taking the ablative-modalis case. Again, notice that the object indicated in the ablative-modalis case becomes an indefinite object. Example (21b) uses *kuve-* 'to spill' from the patientive verb example (17a) in the previous section, repeated here in (21a), and the next example (21d) uses *tuqu-* with the simplex causative suffix 'to kill' from (20b), repeated here in (21c).

(21) a. *mikelngu-um* *saskaq* *kuv-aa*
child-REL.SG cup\ABS.SG spill-IND.3SG>3SG
'the child spilled the cup'

b. *mikelnguq* *saska-mek* *kuv'-i-uq*
child\ABS.SG cup-ABM.SG spill-APS-IND.3SG
'the child spilled a cup'

c. *angute-m* *ungungssiq* *tuqu-t-aa*
man-REL.SG animal\ABS.SG die-A-IND.3SG>3SG
'the man killed the animal'

d. *angun* *tuqu-c-i-uq* *ungungssi-mek*
man\ABS.SG die-A-APS-IND.3SG animal-ABM.SG
'the man killed an animal'

The valency-rearranging suffixes come from the valency-increasing simplex E (applicative) suffixes in Table 2.3 but are used on ditransitive verb stems to rearrange the theme and recipient roles. This suffix will cause secundative ditransitive verb stems to become indirective types and vice versa. Example (22) takes the secundative ditransitive verb *cikir-* 'to give' which inflects the recipient, as shown in (14c), and changes it into an indirective ditransitive verb with inflects the theme.

(22) *angute-m* *ciki-utek-aa* *neqerrluk* *qimugte-mūn*
man-REL.SG give-VVsm-IND.3SG>3SG dryfish\ABS.SG dog-TER.SG
'the man gave the dryfish to the dog' / 'the man is giving the dryfish to the dog'

In summary, verb stem valency and valency changing suffixes have two varying analyses, one from Miyaoka (2012) and the other from Mithun (2000) and Jacobson (2012), where the former analysis focuses on the morphosyntactic roles and the latter analysis on morphological inflection. Mithun (2000) and Jacobson (2012) have more discrete categories for inflections but there are many exceptions to the inflectional categories. Miyaoka (2012) uses zero-derivation suffixes to handle inflectional changes. There are three types of valency changing suffixes: valency-increasing, valency-decreasing, and valency-rearranging. These suffixes have a variety of functions such as causative and antipassive, and the bases these suffixes can attach to range from highly productive to lexically constrained.

In the second edition of the dictionary (Jacobson, 2012), the valency information for bases and derivational suffixes is not easily accessible. A sample of the lexical entries for the base *cikir-* and the derivational suffix *+vkar-* is given in (23). These samples show entries for lexical entries that concern verb valency and the information is not explicitly stated but embedded in the italicized descriptions, example sentences, and English translations. For this reason, and the varying analyses between Miyaoka (2012), and Mithun (2000) and Jacobson (2012) given above, the present version of the morphological analyzer will not be handling verb stem valency and valency changing suffixes. The next subsection will describe how these suffixes attach to bases and the morphophonological rules that apply.

(23) a. **cikir-** to give; to give one something to have or consume; to give one an illness; to give a gift to # cikiraa 'he is giving something to her' / *the object is the recipient; the thing given may be expressed with the ablative-modalis case:* arnam cikiraa mikelnguq akinek 'the woman give the child some money' (*compare* cikiute- *below*); neqkamek cikirnga! 'give me some food!'; camek cikiisqessit? 'what do you want to be given?'; cikiqenglleq vote-arcuutminek 'proxy' (*neologism*); > cikirtur-, cikiun, cikiuteke-

b. **+vkar-** to let, allow, cause, or compel one to V # *used only with bases that end in a vowel; for bases that end in a consonant,* **-cete[1]-** *is used instead; note that with bases that end in* **te***, either postbase may be used; a recent and not universally accepted variant is* **-vkar-** *used on both vowel- and consonant-ending bases; this is a "compound verbal postbase"; for polarity information see* **-ni-** *and* Practical Grammar of . . . Yup'ik (p. 322ff)*; either the embedded verb or the derived verb or both must be transitive;* < PE pb. vkaʁ-

2.4 Morphophonology

Another major component of the morphological analyzer is implementing all of the phonological and morphophonological processes in CAY. These include the base classes and suffix types that define the elaborate morphologically conditioned phonological rules, the phonological rules that apply generally, and the prosodic stress rule that cause phonological processes.

The development of the main framework on the modern grammar and orthography of CAY began in the 1970s from many different researchers at the University of Alaska Fairbanks Alaska Native Language Center. The linguists developed the current orthography alongside a system for showing which phonological rules each affix triggers. From this research, the primary analysis of noun and verb base classes and the morphophonological symbol notation for morpheme boundaries was created. Under this analysis, the morphophonological processes at the morpheme boundaries depend on two factors: the class of the base and the suffix type defined by the morphophonological symbols (Reed et al., 1977). The first section will list the system of classes for noun and verb bases and the next section will be a guide to the morphophonological symbols.

2.4.1 System of Classes for Noun and Verb Bases

CAY noun and verb bases can be classified into a single system of classes defined by the final one to three letters of the base. Table 2.5 (Reed et al., 1977, p.21) provides a summary and a couple of examples for each class.

	Definition	Example
Class I	Base ending in a single prime vowel	*cali-* 'to work'
Class II	Bases ending in two prime vowels	*ui-* 'husband'
Class III	Bases ending in *e* not preceded by *t*	*neqe-* 'fish/food'
Class IV	Bases ending in *te*. This has three subclasses:	
IVa	Bases where a fricative precedes *te*	*inarte-* 'to lie down'
IVb	Bases where a vowel precedes *te*	*elite-* 'to learn'
IVc	Verb bases which are marked by °, "special *te*"	*kiircete-°* 'to be hot'
Class V	Nouns only, bases ending in *r* preceded by one or two vowels, unless marked with an asterisk	*angyar-* 'boat' *qair-* 'wave'
Class VI	All bases ending in a consonant, not in Class V	
	1) all verb bases ending in *g* or *r*	*ayag-* 'to go'
	2) all noun bases ending in *g*	*acag-* 'paternal aunt'
	3) all noun bases ending in *r* marked with an asterisk	*tan'gurrar*-* 'boy'
	4) all noun bases ending in *g* or *r* preceded by an *e*	*ater-* 'name'

Table 2.5: System of Classes for Verb and Noun Bases (Reed et al., 1977, p.21)

Jacobson (1995) does not explicitly use the class numbering system but instead relies on the description of each class or group of classes, such as using the terms "vowel ending bases" for Class I through IV, "consonant ending bases" for Class V and VI, "*te*-ending bases" for Class IV, and "strong" and "weak" bases to describe Classes V and VI (for the % morphophonological symbol described later).

2.4.2 Morphophonological Symbols

The system of morphophonological symbols used to describe the suffix type was first used in Reed et al. 1977 and subsequently adopted by other grammars (Woodbury, 1981; Jacobson, 1995) and the dictionary (Jacobson, 1984, 2012). Each symbol or combination of symbols

used before the suffix indicates a specific phonological operation and which base class(es) it applies to. A full list of morphophonological symbols is given in Appendix A.1. The rest of this section will walk through the list of morphophonological symbols. For each symbol, I will provide a description of the phonological operation, which class it applies to, and a set of examples that illustrate the range of possible combinations.

The first morphophonological symbol is the retaining or adding suffix type indicated with either the plus (*+*) or period (*.*). This is the simplest type where the suffix is attached directly to the base without dropping any final consonants from the base. The period is a special case where the suffix only attaches to vowel ending bases, Class I-IV. The examples in (24) show the use of suffix retaining and adding types with the *+kar* 'future N' and the *+put/.vut* 'our N' suffixes. The *+put/.vut* suffix is a suppletive allomorph where *.vut* is used for vowel ending noun bases and *+put* is used for consonant ending noun bases.[5]

(24) a. angyarkaq
angyar +kar
angyar -kaq
boat -FUT\ABS.3SG
'future boat'

b. angyarput
angyar +put/.vut
angyar -put
boat -ABS.1PL>SG
'our boat'

c. nunavut
nuna +put/.vut
nuna -vut
land -ABS.1PL>SG
'our land'

[5]For single word IGT focusing on the underlying form representation, I use a modified IGT format. The lines of the IGT include: (1) surface form, (2) underlying form, (3) segmented surface form, (4) morphological glossing, (5) free translation. Because these examples are focusing on morphological structure within a single word, I'm using whitespace between morphemes so that I can align the various representations of each morpheme.

The second morphophonological symbol is the dropping suffix type indicated with the minus (-). This suffix drops any final consonant from the base, Class V-VI. The example in (25) shows the use of the dropping type with the final consonant, *r*, being dropped.

(25) angyalleq
angyar -ller
angya *-lleq*
boat -PST\ABS.3SG
'former boat'

There is another dropping suffix type symbol, the double minus (--) used with suffixes that begin with *li*. This symbol indicates that it can drop just the final consonant similar to the single minus symbol, but it additionally indicates that the bases which end in a consonant with a single vowel preceding it optionally have another process where in addition to dropping the base final consonant, it also drops the preceding vowel and the *l* of the suffix. Both forms of the word, from the single minus and double minus processes, are in free variation. This process is shown in (26) where (26a) uses the final consonant dropping only and (26b) uses the additional vowel and *l* dropping.

(26) a. angyaliuq
angyar --li +'(g/t)uq
angya *-li* *-uq*
boat -make -IND.3SG
's/he is making a boat'

b. angyiuq
angyar --li +'(g/t)uq
angy *-i* *-uq*
boat -make -IND.3SG
's/he is making a boat'

c. napaliuq
napa --li +'(g/t)uq
napa *-li* *-uq*
tree -make -IND.3SG
's/he is making a tree'

The third morphophonological symbol is the eliding or *e*-dropping suffix type indicated with the tilde (~). This suffix drops all base final *e* when attaching to Class III-IV bases. The example in (27) shows the use of the suffix *~+miu* 'resident of N' which also includes the plus symbol to indicate how it handles consonant ending bases.

(27) a. akulmiu
akule ~+miu
akul *-miu*
area.between -resident\ABS.3SG
'resident of the (land) in between'

b. nunamiu
nuna ~+miu
nuna *-miu*
land -resident\ABS.3SG
'resident of the land'

The fourth morphophonological symbol is the half-retaining suffix type indicated with the percent (%). This suffix drops the final *r* from Class V bases (that is weak bases), but keeps the final consonant on Class VI bases (that is strong bases). The examples in (28) show the use of the *%~mek* 'ABM.3SG' suffix attaching to a weak base (28a) and to strong bases (28b, 28c).

(28) a. angyamek
angyar %~mek
angya *-mek*
boat -ABM.3SG
'a boat'

b. kuigmek
kuig %~mek
kuig *-mek*
river -ABM.3SG
'a river'

c. nayirmek
nayir* %~mek
nayir *-mek*
seal -ABM.3SG
'a ringed seal'

The fifth morphophonological symbol is the velar-uvular dropping suffix type indicated with the colon (*:*). This suffix indicates a pattern where a velar or uvular consonant (*g*, *r*, *ng*) at the morpheme boundary is dropped if that consonant is flanked by single vowels. This process also causes vowel raising and fronting with certain vowel and consonant combinations, as well as vowel assimilation when an *e* occurs next to a full vowel. The examples in (29) show the use of the *%:(e)t* 'ABS.3PL' suffix which includes the percent symbol for half-retaining and *e* in parentheses (explained in the section below) for adding the letter in consonant ending bases.

(29) a. angyat
angyar %:(e)t
angya *-t*
boat -ABS.3PL
'boats'

b. kuiget
kuig %:(e)t
kuig *-et*
river -ABS.3PL
'rivers'

c. nayiit
nayir* %:(e)t
nayi *-it*
seal -ABS.3PL
'ringed seals'

d. atkuut
atkug %:(e)t
atku *-ut*
parka -ABS.3PL
'parkas'

Example (29a) shows that velar/uvular consonant dropping takes priority over *e*-insertion. Example (29b) shows that the base final *g* does not get dropped because there are two vowels on the left-side of the consonant. Example (29c, 29d) shows the *(e)* gets added and the base final consonant is dropped from the flanking single vowels as well as the additional vowel assimilation process where the e assimilates to the preceding vowel.

The sixth morphophonological symbol is the short base gemination suffix type indicated by the apostrophe (’). This suffix causes the consonant, C_2, of short bases, defined as having the form #(C_1)VC_2*e*-, to be geminated as indicated by an apostrophe in the orthography. Example (30) shows the suffix *+’(g)aqa* ‘IND.1SG>SG’ attaching to the short base *nere-* ‘to eat’ causing the *r* consonant to be geminated as indicated by the apostrophe.

(30) ner’aqa
nere +’(g)aqa
ner’ -aqa
eat -IND.1SG>3SG
‘I am eating it.’

The seventh morphophonological symbol is the *te* affecting suffix type as indicated by the at sign (@). A suffix marked with @ indicates that the *te*-ending bases (Class IV) are treated differently compared to the other base classes. The @ type suffix drops the *e* following the *t* and a summary of how the *t* changes is shown in Table 2.6 (Reed et al., 1977, p.29). The first letters of the suffix indicate what suffix type it is and the Ø sign indicates the *t* is dropped. Since the suffix type and its associated changes is predictable from context, Jacobson (1995) does not implement the suffix type superscript notation (e.g. $@^1$).

Example set (31) uses the postbase *@~+ngaite* ‘FUT.NEG’ and which is subtype $@^2$ using bases from each of the Class IV subclasses.

(31) a. ceñirrngaituq
ceñirte @~+ngaite +’(g/t)uq
ceñirr -ngait -uq
visit -FUT.NEG -IND.3SG
‘s/he won’t visit’

suffix starts with:	type:	IVa	IVb	IVc
n	@[1]	t → Ø and resulting cluster voiceless	t → t	
ng, m, v (postbases)	@[2]		t → s	t → l
(u)	@[3]	t → Ø	t → y	
ng, g, k (endings)	@[4]	t → s		
c, p	@[5]	t → Ø		

Table 2.6: Summary of Different *te* Affecting Patterns (Reed et al., 1977, p.29)

b. kipusngaituq
kipute @~+ngaite +'(g/t)uq
kipus *-ngait* *-uq*
visit -FUT.NEG -IND.3SG
's/he won't buy (something)'

c. assiilngaituq
assiite° @~+ngaite +'(g/t)uq
assiil *-ngait* *-uq*
be.bad -FUT.NEG -IND.3SG
'it won't be bad'

Another major set of symbols are the letters in parentheses. Certain suffixes start with a letter in parentheses and these letters are used with some of the base classes and not the others. The most common letters in parentheses are given in Table 2.7 (Reed et al., 1977) with the full table of letters in parentheses is listed in Appendix A.1.

(g)	is used with bases ending in two vowels	(Class II)
(ng)	is used with bases ending in a vowel	(Class I-IV)
(s)	is used with bases ending in a vowel	(Class I-IV)
(t)	is used with bases ending in a consonant	(Class V-VI)
(u)	is used with bases ending in a consonant or *e* (includes *te*)	(Class III-VI)

Table 2.7: Letters in Parentheses

The set of examples in (32) use the IND.3SG suffix, *+'(g/t)uq*, which has two different letters in parentheses, *(g)* and *(t)*. Example (32b) uses the *(g)* because the base ends in two

vowels and Example (32c) uses the *(t)* because the base ends in a consonant.

(32) a. caliuq
cali +'(g/t)uq
cali -uq
work -IND.3SG
's/he is working'

b. qiaguq
qia +'(g/t)uq
qia -guq
cry -IND.3SG
's/he is crying'

c. qavartuq
qavar +'(g/t)uq
qavar -tuq
sleep -IND.3SG
's/he is sleeping'

The underline diacritic on the first letter of suffixes is not in use in Jacobson's (1995) grammar book and the second edition of the dictionary (Jacobson, 2012) but was included in Reed et al.'s (1977) grammar book and first edition of the dictionary (Jacobson, 1984). This diacritic was used on suffixes that begin with a velar or uvular consonant (i.e. *k*, *q*, *g*, *r*, *gg*, or *rr*) to indicate the assimilating type of dropping suffix. These suffixes are marked with the minus symbol to drop the base final consonants, but, in addition, if the consonant is a velar consonant and the suffix begins with a uvular consonant or vice versa, then the underlined suffix will assimilate to the corresponding place of articulation of the base final consonant. Since this morphophonological process occurs with the majority of suffixes that begin with a minus symbol and a velar/uvular consonant and the exceptions[6] to this rule are in the minority, this diacritic was removed from Jacobson's later publications. Example (33) show the assimilating suffix -*ka* 'ABS.1SG>3SG' with two different bases, one ending in a velar suffix (same place of articulation as the suffix) and one ending in a uvular suffix

[6]The exceptions to the rule are described by Jacobson in the definitions of lexical entries instead.

(differing place of articulation). As seen in Example (33b), when the velar suffix attaches to the uvular ending base, the *k* of the suffix assimilates to the uvular place of articulation.

(33) a. panika
panig -ka
pani *-ka*
daughter -ABS.1SG>3SG
'my daughter'

b. qayaqa
qayar -ka
qaya *-qa*
kayak -ABS.1SG>3SG
'my kayak'

2.4.3 Other Morphophonological Processes

In addition to the set of morphologically conditioned phonological rules indicated by the morphophonological symbols, there are other regular morphophonological processes that occur in specific environments.

A major process in CAY is the devoicing of fricatives and nasal consonants next to stops and voiceless fricatives. This devoicing process is not reflected in the orthography where the devoiced consonants are still written as their voiced counterparts as explained in Rule 1 in Section 2.2.1. A devoicing process whose output is represented in the orthography is when *y* becomes a voiceless *s* when next to stops and voiceless fricatives as shown in Example (34).

(34) a. taqsugtuq
taqe @~+yug +'(g/t)uq
taq *-sug* *-tuq*
quit -DES -IND.3SG
's/he wants to quit'

b. aurrsugtuq
aurre @~+yug +'(g/t)uq
aurr -sug -tuq
crawl -DES -IND.3SG
's/he wants to crawl'

It is a feature in CAY that triple consonant clusters are not permitted within the word. When a suffix is added that results in prohibited consonant clusters, an *e* is inserted to break up the cluster, either between the first and second consonants or the second and third consonants. In Example (35), the base after attachment of the first suffix results in a triple consonant cluster, **mingqsug-*. To break the cluster, an *e* is either inserted between the first two consonants as in (35a) or between the second two consonants as in (35b). Both forms are in free variation but certain combinations of triple consonant clusters choose where they break up the cluster such as inserting an *e* following a middle consonant *t* or preceding a middle consonant *s*.

(35) a. mingeqsugtuq
mingqe @~+yug +'(g/t)uq
mingeq -sug -tuq
sew -DES -IND.3SG
's/he wants to sew'

b. mingqessugtuq
mingqe @~+yug +'(g/t)uq
mingqe -ssug -tuq
sew -DES -IND.3SG
's/he wants to sew'

In certain base and suffix combinations, triple vowel clusters arise underlyingly. Triple vowel clusters are not permitted within the word, excluding enclitics. When a three vowel cluster occurs, the middle vowel is dropped. In Example (36), the base attaching to the inflectional suffix results in a triple vowel cluster, **nalluai*, and so by this rule, the middle vowel is dropped.

(36) nallui
nallu +'(g)ai
nallu *-i*
not.know -IND.3SG>3PL
's/he doesn't know them.'

In CAY, at the ends of words or enclitic boundaries there are a set of word final rules. These rules change *r* to *q*, *g* to *k*, V*te* to V*n*, and *e* to *a* (after the previous rule is applied). A set of examples demonstrating these rules are given in (37).

(37) a. qayaq
qayar
qayaq
kayak\ABS.SG
'a kayak'

b. kuik-llu
kuig =llu
kuik *=llu*
river\ABS.SG =and
'and a river'

c. cavun
cavute
cavun
oar\ABS.SG
'an oar'

d. qimugta
qimugte
qimugta
dog\ABS.SG
'a dog'

Another rule has *ti* become *ci* at morpheme boundaries. In Example (38), the double minus suffix drops the letters until the base becomes **akuti-* and then the rule changes it to *akuci-* 'to make *akutaq* (a dish traditionally made from a mixture of whipped caribou fat or seal oil and berries)'.

(38) akuciuq
akutar --li +'(g/t)uq
akuc *-i* *-uq*
akutaq -make -IND.3SG>3PL
's/he is making *akutaq*'

Yet another rule has *qar* become *qer* if the syllable is at a morpheme boundary such as in (39).

(39) apteqerru
apte -qar -ggu
apte *-qe* *-rru*
ask -please -OPT.2SG>3PL
'please ask him/her'

2.4.4 Prosodic Adjustment Rules

Prosody in CAY includes rhythmic lengthening of vowels, automatic gemination, and rhythmic stress as explained in Section 2.2.1. A common rule that arises from prosody is that an *e* that would be rhythmically lengthened (also known as hatted *e*, i.e. *ê*) is dropped, unless it is between identical or similar consonants. It is dropped because the reduced vowel *e* cannot be lengthened. This rule has an additional feature where a voiced fricative stays voiced even when it follows a stop or voiceless fricative and is indicated in the orthography with an apostrophe separating the two consonants. However, a voiced fricative becomes voiceless when it precedes a stop or voiceless fricative in GCY although it stays voiced in other CAY dialects like HBC. An example set of these processes is given in (40).

All of the examples in (40) have rhythmic lengthening on the final *e* of the base that is considered for hatted *e* dropping. Example (40a) drops the hatted *e* from *tumêlek* with no other processes. Example (40b) does not drop the hatted *e* from *tumêmek* because it is between two identical consonants. Example (40c) drops the hatted *e* from *tepêlek* but also preserves the voiced consonant, *l*, following the stop consonant, *p*, with an apostrophe.

Example (40d) and (40e) drop the hatted *e* from *nerêciquq* but GCY does not preserve the voiced *r* preceding the stop consonant, *c*, whereas HBC does preserve the voicing.

(40) a. tumlek
tume -leg
tum *-lek*
footprint -one.withABS.SG
'one having a footprint'

b. tumemek
tume %~mek
tume *-mek*
footprint -ABM.SG
'footprint'

c. tep'lek
tepe –leg
tep' *-lek*
odor -one.withABS.SG
'one having an odor'

d. nerciquq
nere +ciqe +'(g/t)uq
ner *-ciq* *-uq*
eat -FUT -IND.3SG
's/he will eat' (GCY)

e. ner'ciquq
nere +ciqe +'(g/t)uq
ner' *-ciq* *-uq*
eat -FUT -IND.3SG
's/he will eat' (HBC)

A select few bases begin with something called a weak initial *e* indicated with an *e* in square brackets *[e]*. A weak *[e]* always occurs in word-initial syllables. This weak *e* is kept when it is needed for the word to not become monosyllabic or to preserve the phonological context that allows the orthographic automatic gemination rule to apply. The weak *e* is dropped in all other cases, but when the base begins with a weak *e*, a consonant, a

rhythmically lengthened *e*, and another consonant (i.e. */e/*$C_1\hat{e}C_2$), then the second consonant receives an apostrophe to show marked gemination, since the rhythmically lengthened *e* loses its lengthening when the weak *e* is dropped (Reed et al., 1977). An example set for each of these cases is given in (41) using the base */e/na*.

(41) a. ena
[e]ne
ena
house\ABS.SG
'a house'

b. enii
[e]ne :(ng)a
eni *-i*
house -ABS.3SG>SG
'his/her house'

c. nerpak
[e]ne –rpag
ne *-rpak*
house -big\ABS.SG
'a big house'

d. nek'a
[e]ne –ka
ne *-k'a*
house -ABS.1SG>SG
'my house'

Examples (41a) and (41b) both keep the weak *e*, the former to keep the word from becoming monosyllabic and the latter to preserve the phonological context of the automatic gemination rule on the *n*. The weak *e* in (41c) is dropped because it does not apply to the previous two rules. Example (41d) also drops the weak *e* but the effect of the weak *e* is represented in the gemination of the second consonant from the rhythmic lengthening of the middle *e*. Thus */e/nêka* becomes *nek'a*.

Now the final morphophonological process that utilizes the prosody is the process called

ar-deletion indicated with the *ar* in parentheses *(ar)*, or *(ar*)* when used with nominal bases and suffixes. The *(ar)* indicates that this segment can be optionally deleted if the *ar* is followed by a consonant or the end of the word. In the examples shown in (42), the *ar* in the suffix is deleted because it either appears at the end of the word (42a) or followed by a consonant (42b), but the *ar* is not deleted in (42c) because the *r* was dropped by the following suffix.

(42) a. qayacuar
qayar –cuar(ar*)
qaya *-cuar*
kayak -small\ABS.SG
'a small kayak'

b. qayacuarkaq
qayar –cuar(ar*) +kar
qaya *-cuar* *-kaq*
kayak -small -FUT\ABS.SG
'a future small kayak'

c. qayacuaralleq
qayar –cuar(ar*) -ller
qaya *-cuara* *-lleq*
kayak -small -PST\ABS.SG
'a former small kayak'

Now the examples in (42) did not behave unexpectedly because the preceding syllable *cua* was stressed and the *rar* syllable with the *ar* was not stressed. However, when the preceding syllable is rhythmically lengthened before *ar*-deletion or when the syllable with the *ar* is stressed, then the prosody of the word is affected and certain processes occur to accommodate the stress changes. The two examples in (43) demonstrate these processes.

The first example (43a) has the stress pattern *úm.yuár.te.qû.rar.tuq* before *ar*-deletion. When the *ar* is deleted, the resulting word is *úm.yuár.te.qûr.tuq* where the *û* retains its rhythmic lengthening, but this lengthening is no longer following the rhythmic lengthening rule since it is part of the closed syllable. To indicate that the *û* is lengthened, we instead

use two *u* with an apostrophe in between, *u'u*, to show it is lengthened and blocks automatic gemination of the *q*.

The second example (43b) has the stress pattern *íq.vár.tu.rár.tuq* before *ar*-deletion. When *ar* is deleted, the stress on the deleted *ar* is shifted to the preceding syllable in what Jacobson (1995) calls ghost stress. This results in the word *íq.vár.túr.tuq* which does not follow the rules for rhythmic stress. Ghost stress is not always explicitly indicated in writing, but can be indicated with an apostrophe between the vowel and the following consonant.

(43) a. umyuartequ'urtuq

umyuarteqe	@ˤ+'(g/t)ur(ar)	+'(g/t)uq
umyuarteq	*-u'ur*	*-tuq*
think	-continue.to	-IND.3SG

's/he keeps on thinking'

b. iqvartu'rtuq

iqvar	@ˤ+'(g/t)ur(ar)	+'(g/t)uq
iqvar	*-tu'r*	*-tuq*
pick.berries	-continue.to	-IND.3SG

's/he keeps on picking berries'

2.5 Summary

This chapter presented an overview of the necessary background literature of CAY and morphology for this project. It provided a summary of the major works in CAY grammar literature, an introduction to the phonetic inventory and orthographic rules, the morphotactics of verbs and nouns including derivational suffixes and valency changing suffixes, and the system of morphophonological symbols and processes. The next chapter will present an overview of the finite-state morphological tools used for this project.

Chapter 3

FINITE-STATE MORPHOLOGY

This chapter will provide an overview of finite-state transducers (FSTs) and their use as morphological analyzers. Finite-state morphological analyzers are a popular technology used to model the morphology of natural languages.

The serious development of the use of finite-state transducers for morphology began in the 1980s with a comprehensive work published in Kaplan and Kay 1994 (although developed much earlier in Kaplan and Kay 1981) and Koskenniemi 1983. A lot of work went into the development of automatic rule compilers and algorithms for manipulating finite-state networks at Xerox/PARC (Karttunen et al., 1987; Karttunen and Beesley, 1992; Karttunen, 1993; Beesley and Karttunen, 2003a). Two good introductory resources into finite-state morphology are *Finite State Morphology* and *Two-level Rule Compiler* by Beesley and Karttunen (2003a,b).

Section 3.1 provides a quick introduction to FSTs in their relation to morphological analyzers. Section 3.2 will describe the three formalisms used to create FSTs and the popular toolkits used to manipulate FSTs. Section 3.3 will give a survey of other languages that have created finite-state morphological analyzers and finally, Section 3.4 will look at the uses for finite-state morphological analyzers in the creation of other language technologies and resources.

3.1 Finite-State Transducers

A finite-state transducer (FST) is an abstract machine that can be used to model the morphology of natural languages. An FST is made up of states and arcs which have an input symbol and output symbol as shown in Figure 3.1. This FST, a lexical transducer, takes the

Spanish word *canto* input and outputs the underlying form of the word.

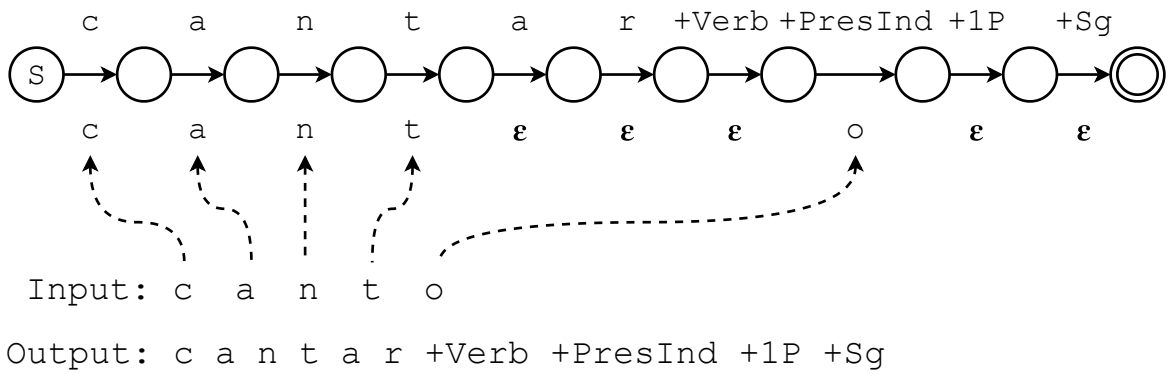

Figure 3.1: Finite-State Transducer Analyzing *canto* (Beesley and Karttunen, 2003a, p.13)

In Figure 3.1, the set of states are indicated by the circles and the arcs by arrows between the states. This FST shows a path of states and arcs that take the Spanish word *canto* one symbol at a time and outputs the underlying form one symbol at a time. This FST begins at the start state indicated by the `S` and matches the input symbol on the arcs, consuming the input symbol, and follows the path to the final state indicated by two circles. When it reaches the final state, it outputs the string of output symbols from the path it followed . Notice how the symbols may be multi-character such as `+Verb` and still be considered one symbol, or they may be epsilon arcs indicated by the ε (or `0`) which represent arcs that may be traversed freely in the input or the empty string in the output (Beesley and Karttunen, 2003a).

FSTs have many properties that allow them to be extremely flexible and efficient at pattern matching and string translation tasks. The first beneficial property is that FSTs are bidirectional, meaning they can be easily inverted and the input becomes the output and the output the input. This means that lexical transducers such as the one shown above can perform morphological analysis, going from surface form to underlying form, or morphological generation, going from underlying form to surface form. The underlying form

can be phonological (underlying phonemes) or morphological (morphemes + tags), according to how the linguist designs the FST (Beesley and Karttunen, 2003a).

Linguists traditionally describe the morphology and phonology of word-formation in the generation direction. They start with an abstract underlying form and apply a set of morphological and phonological conditional rules to arrive at the surface form of the word. These rules include elision, epenthesis, assimilation, devoicing, vowel lengthening, and gemination to sounds in certain environments. Given the FST compilation techniques of Karttunen et al. 1987; Karttunen and Beesley 1992; Karttunen 1993; Beesley and Karttunen 2003a, these descriptive grammars may be used to construct a lexical transducer in the generation direction and then invert it to provide a morphological analyzer (Karttunen et al., 1992; Karttunen, 1994).

The second beneficial property is composition and modularity, meaning all FSTs can handle different modules being composed together to create a single FST. Composition allows many separate FSTs built and stacked together end to end to create a single lexical transducer. Modularity means there are separate transducers that handle lexical and morphological and phonological phenomena. Figure 3.2 shows a common FST stack to create a lexical transducer.

The basic FST stack, shown in Figure 3.2, is comprised of two modules, the lexicon transducer and the rule transducer, composed together to create a single lexical transducer. The lexicon transducer handles all of the lexical entries and morphotactics that map the underlying form to an intermediate form. The rule transducer may be composed of all of the separate morphological and phonological rule FSTs necessary to map the intermediate form to the surface form (Karttunen, 1994).

The third beneficial property is the small size and efficiency of FSTs. A lot of work has gone into making efficient compilers and algorithms for manipulating FSTs and this has resulted in the availability of large finite-state morphological analyzers that compile quickly, do not take up a lot of disk space, and still analyzes words quickly (Karttunen, 1994). The next section describes and compares the three formalisms used to compile FSTs.

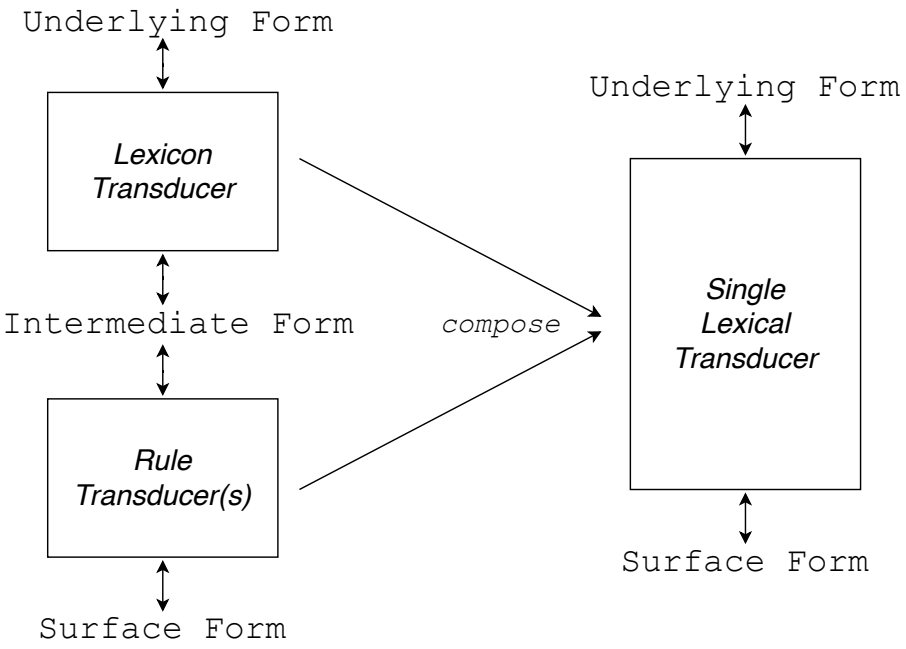

Figure 3.2: FST Stack for a Finite-State Morphological Analyzer (Beesley and Karttunen, 2003a, p.35)

3.2 FST Compilation Formalisms

Instead of creating finite-state transducers directly by describing explicit states and arcs, linguists using FSTs to model morphology typically create them by using compilation formalisms, high-level declarative languages which compile into an FST. The three common formalisms are *lexc* (Karttunen, 1993), *xfst* (Karttunen, 1994, 1995; Beesley and Karttunen, 2003a), and *twolc* (Koskenniemi, 1983) as indicated by Figure 3.3. The *lexc* formalism is used to create the lexicon transducer which handles the lexical entries and morphotactics of a language. The *xfst* and *twolc* formalisms are used to create rule transducers that handle the morphology and phonology of the language.

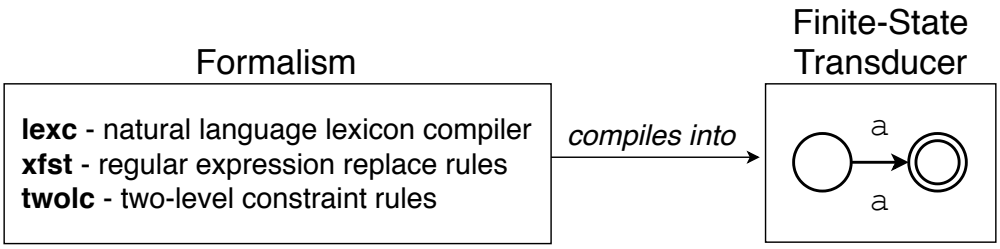

Figure 3.3: Finite-State Transducer Compilation Formalisms

3.2.1 lexc

The *lexc* formalism (from Lexicon Compiler) (Karttunen, 1993) is well suited to describe the lexicon of the language which include the lexical entries and the morphotactics, the ordering restriction of morphemes in word formation. Although it might not handle non-concatenative morphology well (see Beesley and Karttunen 2003a, p.375), it is well suited for most concatenative morphologies. The formalism uses the idea of breaking the words into their morphemes and defining classes of morphemes into their own sublexicon. This formalism defines sets of sublexicons which are usually sets of morpheme classes, the lexical entries in that sublexicon, followed by the continuation class, the name of the next sublexicon which concatenatively follow that entry (Karttunen, 1993). An example of a sublexicon is given (44).

(44)
```
LEXICON Name
entry1        NextLexiconName ;
...
entryN        NextLexiconName ;
```

There are two special lexicons for defining the first sublexicon `Root`, that begins the word in concatenative order, and the last sublexicon `#` that ends the word and has no lexical entries. Empty lexical entries are allowed which indicate epsilon arcs to the next sublexicon in concatenative order. Another feature is that lexical entries may use the colon (`:`) to

indicate the separate input and output strings when they are not one-to-one. Examples of these features are given in a toy example of English words in (45).

(45)
```
LEXICON Root
        Nouns ;
        Verbs ;

LEXICON Nouns
dog     N ;
moose   # ;

LEXICON N
        # ;
s       # ;

LEXICON Verbs
walk      V ;
swim:swam # ;

LEXICON V
        # ;
s       # ;
ed      # ;
ing     # ;
```

This toy example of English words in (45) begins words starting at the `Root` sublexicon which has empty lexical entries that continue into the `Nouns` and `Verbs` continuation classes. The `Nouns` sublexicon has two entries, `dog` which goes onto the `N` sublexicon to attach an optional plural marker, and `moose` which goes directly to end the word because it does not inflect with the plural marker. This branch creates the words {`dog, dogs, moose`}. The `Verbs` sublexicon has two entries, `walk` which goes onto the `V` sublexicon to attach inflection, and the lexical entry `swim:swam` which defines the input string `swim` which has an irregular past tense output string `swam`. This branch creates the words {`walk, walks, walked, walking, swim:swam`}. Notice that `swims` is not permitted because `swim` does not go to the `V` continuation class.

3.2.2 xfst

The *xfst* formalism (Karttunen, 1994, 1995; Beesley and Karttunen, 2003a) is one of two formalisms used to describe the morphological and phonological rules of the language. The *xfst* formalism is based off a regular expression notation with an extension including replace rules. These rules mimic the rewrite rule formalism by Chomsky and Halle (1968) and are of the form:

(46) `a -> b || c _ d`

which indicates that the string `a` in the upper-side is replaced by the string `b` in the lower-side in the context where `a` occurs between `c` in the left-hand context and `d` in the right-hand context.

If there are multiple replace rules, these rules are placed in series where the output of the first rule is the input for the second rule, therefore, the linguist must keep track of the rule order. Example (47) from Beesley and Karttunen 2003a describes two rules for a fictional language where `kaNpat` an abstract lexical string consists of the morpheme `kaN` (containing an underspecified nasal morphophoneme `N`) concatenated with the suffix `pat`.

(47)

Lexical:	`kaNpat`
	`N -> m || _ p`
Intermediate:	`kampat`
	`p -> m || m _`
Surface:	`kammat`

The first replace rule in (47) takes the underspecified nasal morphophoneme `N` just before a `p` and replaces it with an `m` which yeilds an intermediate form `kampat`. Then the second replace rule takes the `p` that occurs after an `m` and replaces it with an `m` to get the surface form `kammat`.

Since the replace rules are in series, they can be compiled and composed together to create a single rule transducer that directly maps the abstract lexical form into the surface

form. Then there is another composition of the lexical transducer and the rule transducer to create the composite FST as seen in Figure 3.4(a) and 3.4(b).

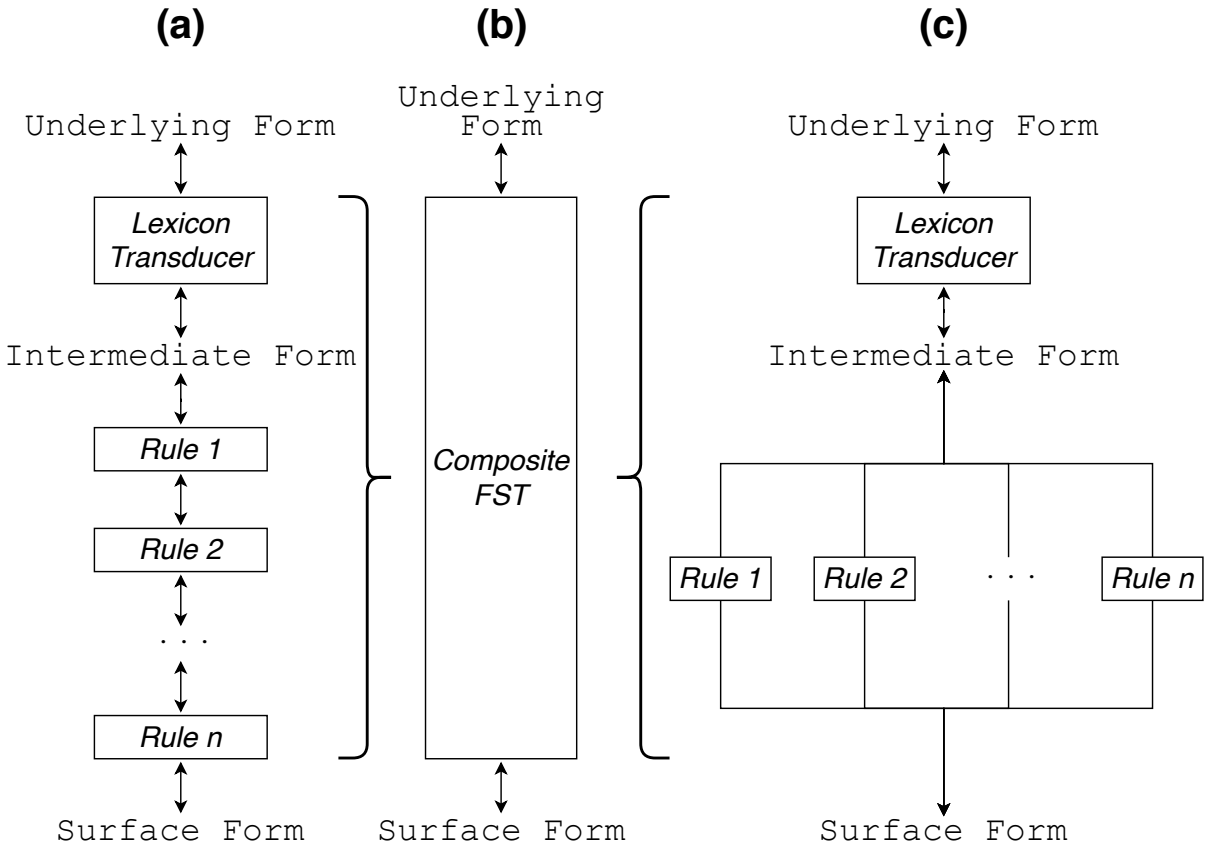

Figure 3.4: (a) a series of *xfst* rewrite rules composed into, or (c) *twolc* constraint rules apply in parallel and compose to (b) a single equivalent composed FST (Hulden, 2009a, p.29)

3.2.3 twolc

The *twolc* formalism (from Two-Level Compiler) is the other formalism used to describe the morphological and phonological rules of the language. The syntax and semantics of writing grammar rules in the *twolc* formalism require a different mindset. The first difference is that instead of *xfst* replace rules being applied sequentially, *twolc* constraint rules are applied in

parallel (as seen in Figure 3.4(c)). The next difference is these symbol-to-symbol constraints can refer to the upper-level context, to the lower-level context, or to both upper and lower contexts at the same time. Another major difference is that the two levels require an equal number of symbols in the upper-level and lower-level. This means that certain symbols that get deleted will be paired with a `0` so that each string always has exactly the same length, and after the rules are applied, the `0` will be treated as an empty string (Beesley and Karttunen, 2003b). An example of a constraint rule is given in (48):

(48)
```
"Unique Rule Name"
a:b <=> c _ d
```

which indicates that the symbol `a` is realized as `b` in between `c` in the left-hand context and `d` in the right-hand context and nowhere else.

Using the same example from (47) above, Example (49) describes the *twolc* constraint rules to implement the `kaNpat` to `kammat`. Notice that the two strings are aligned with an equal number of symbols and each rule is constraining a symbol pair.

(49)

Lexical:	`k a N p a t`	`k a N p a t`
	`: : | : : :`	`: : : | : :`
Surface:	`k a m m a t`	`k a m m a t`
	`"N:m rule"`	`"p:m rule"`
	`N:m <=> _ p:`	`p:m <=> :m _`

The first replace rule in (49) constrains the `N:m` pair to occurring only next to the `p` in the upper-level right-hand context and the second rule constrains the `p:m` pair in the lower-level left-hand context. These two constraint rules are applied in parallel and are compose-intersected[1] with the lexical transducer resulting in a single composite FST as shown in Figure 3.4(b) and 3.4(c).

[1]Since *twolc* rules are applied in parallel, "the intersecting composition algorithm performs the intersection and composition of the rules simultaneously" (Beesley and Karttunen, 2003b).

An interesting fact is that a series of *xfst* replace rules or a set of parallel *twolc* constraint rules, can in principle always be combined into an equivalent single transducer. Replace rules are merged by composition and constraint rules by intersection. The final transducer can be the same. The two formalisms represent different intuitions of how to accomplish the task. Replace rules work well when there is a large difference in the upper string to the lower string such as when there are many deletions and insertions. Constraint rules on the other hand are well suited to model processes where there is an exact symbol-to-symbol correspondence or when rules reference both the upper-level and lower-level contexts Beesley and Karttunen (2003b).

3.2.4 FST Toolkits

There are many FST toolkits, software and libraries used to compile and manipulate FSTs, used to create morphological analyzers such as:

(50) a. Xerox/PARC Finite State Morphology Tools[2] (Beesley and Karttunen, 2003a)
b. Stuttgart Finite State Transducer (SFST) tools[3] (Schmid, 2005)
c. Foma[4] (Hulden, 2009b)
d. Helsinki Finite-State Technology[5] (HFST) (Lindén et al., 2011)

The Xerox/PARC line of tools accompanies the *Finite State Morphology* book (Beesley and Karttunen, 2003a) and has compilers for the *xfst* formalism of regular expression replace rules, the *lexc* formalism for the lexicon transducer, and for Koskenniemi's *twolc* formalism, along with the tools to manipulate FSTs. It is fast in FST lookup functionality and fast in compiling lexicon and rules, but it is closed source, requiring users to get a license for the source code.

---

[2] https://web.stanford.edu/~laurik/fsmbook/home.html

[3] https://www.cis.uni-muenchen.de/~schmid/tools/SFST/

[4] https://fomafst.github.io

[5] https://hfst.github.io

The SFST toolkit (Schmid, 2005) was the first open source FST toolkit and SFST presented its own formalism called SFST-PL to compile and manipulate FSTs. This formalism is based on extended regular expressions. Other open source toolkits have been used recently in favor of this toolkit.

The foma toolkit (Hulden, 2009b) is an open source program that is compatible with the Xerox/PARC toolkit. It includes a compiler for the *xfst* and *lexc* formalism, and a C library for constructing FSTs. Hulden (2009b) has also added the ability to work with multi-tape finite-state transducers and a new formalism for expressing first-order logical constraints. This toolkit is also fast in lookup and fast in compilation, but does not include support for the *twolc* formalism.

The HFST toolkit (Lindén et al., 2011) is an open source program that is compatible with the Xerox/PARC toolkit and also compatible with the SFST-PL formalism as well. This toolkit is unique in that it connects the backend to allow the multiple existing open source libraries for compiling and manipulating FSTs. These libraries include SFST, foma, and OpenFst (Allauzen et al., 2007), which also allows weighted FSTs. HFST also implemented its own compilers for the four formalisms and other FST manipulating tools with the ability to store FSTs into the multiple backend formats. This toolkit is fast at lookup but slower at compiling compared to the other tools, but it has the ability to support weighted FSTs.

For my project, I used the HFST toolkit because it is open source and has the ability to compile the three major formalisms. That way I had the ability to choose which formalism best suited each morphophonological phenomenon. It also has the ability to support weighted FSTs which can possibly be used in the future as a way to add parse disambiguation.

3.3 Languages with Morphological Analyzers

In this section I will be looking at other agglutinative and polysynthetic languages with morphological analyzers, with a focus on the Inuit-Yupik-Unangan (IYU) language family.

3.3.1 Inuit-Yupik-Unangan Language Family

The first morphological analyzer that was developed for an IYU language was for Kalaallisut (West Greenlandic) (ISO 639-3: kal) by Oqaasileriffik through the help of the Divvun and Giellatekno groups at the University of Tromsø (Oqaasileriffik, 2010). The Divvun and Giellatekno language technology (GiellaLT) infrastructure, first developed for the Sámi languages, is hosting the development of finite-state morphological analyzers of Kalallisut, Iñupiaq (ISO 639-3: ipk) (Bills et al., 2010), and Inuktitut (ISO 639-3: iku). This infrastructure supports the development of morphological analyzers using the Xerox/PARC, HFST, and foma toolkits. Since 2013, the Kalallisut analyzer has been in production development which means it has good coverage (72.4%), complete grammatical description, and is being used for end user tools. The Iñupiaq analyzer was still in a development status which means the morphological description is well on its way and the lexicon is substantial. The Inuktitut analyzer was still getting started with some initial work being done, but the lexical content is very limited (Moshagen et al., 2013).

Another IYU language morphological analyzer is for Inuktitut (ISO 639-3: ipk) developed by the Institute for Information Technology of the National Research Council of Canada but instead of using FSTs, it is written in Java. This morphological analyzer is reported to analyze over 95% of the most frequent words found in the Nunavut's Hansard [corpus] and Inuktitut Web pages (Institute for Information Technology, 2012).

The first morphological analyzer to be developed for a Yupik language was for St. Lawrence Island/Central Siberian Yupik (CSY) (ISO 639-3: ess). This analyzer is implemented in foma using the *lexc* and *xfst* formalisms. Chen and Schwartz (2018) report that the first version of the morphological analyzer has around 97% precision and recall over a corpus of end-of-chapter exercises from their reference grammar. Chen et al. (2020) report that the second version of the morphological analyzer, using a different theory of the morphology, is only partially implemented and has around 99% precision and 94% recall over the end-of-chapter exercises, and a coverage of 12% of types and 21% of tokens over a corpus

of text.

The morphological analyzer I have created for CAY builds on the underlying form representation that Chen and Schwartz (2018) has created for CSY. Since the grammar and morphology of CAY is very similar to CSY, I did not have to develop my own underlying representation. A possible advantage for having a unified underlying form is the ability to easily swap out the morphological analyzer with applications that are built for either language (see Section 3.4). Another possible advantage is that it may facilitate machine translation between the CAY and CSY by a lexical transfer dictionary.

3.3.2 Other Agglutinative and Polysynthetic Languages with Morphological Analyzers

Language	**ISO 639-3**	**Toolkit**	**Formalism**	**Reference**
Cayuga	cay	Xerox/PARC	*xfst*	Graham 2007
Finnish	fin	HFST		Lindén and Pirinen 2009
Turkish	tur	SFST & foma	*xfst*	Çöltekin 2010
Mohawk	moh	Xerox/PARC	*xfst*	Assini 2013
North Saami	sme	GiellaLT		Johnson et al. 2013
Lakota	lkt	Xerox/PARC	*xfst*	Curtis 2014
Plains Cree	crk	GiellaLT	*twolc*	Snoek et al. 2014; Harrigan et al. 2017
Cuzco Quechua	quz	Xerox/PARC	*xfst*	Rios 2015
Tsuut'ina	srs	GiellaLT	*xfst/twolc*	Arppe et al. 2017a
East Cree	crl	GiellaLT	*xfst/twolc*	Arppe et al. 2017b
Odawa	otw	GiellaLT	*xfst*	Bowers et al. 2017
Arapaho	arp	foma	*xfst*	Kazeminejad et al. 2017
Bribri	bzd	foma	*xfst*	Solórzano 2017
Northern Haida	hdn	GiellaLT	*twolc*	Lachler et al. 2018
Chukchi	ckt	HFST	*twolc*	Andriyanets and Tyers 2018
Upper Tanana	tau	GiellaLT	*xfst*	Lovick et al. 2018

Table 3.1: Other Agglutinative or Polysynthetic Finite-State Morphological Analyzers

In addition to the IYU languages, there are many other finite-state morphological analyzers for agglutinative or polysynthetic languages listed in Table 3.1. These analyzers use a variety of toolkits including the Giellatekno language technology (GiellaLT) infrastructure

which includes Xerox/PARC, foma, and HFST toolkits. They also use a mixture of rule transducer formalisms.

3.4 Uses for Analyzers

Once a finite-state morphological analyzer has been developed for a language, the analyzer can be integrated with other applications. The first application is an intelligent electronic dictionary. Since a lexicon was necessary for a high coverage analyzer, the lexical database can be combined with the morphological analyzer to allow users to search fully inflected forms in the dictionary and return a parse where the user can click on relevant lexical entries and see the individual definitions. Utilizing the morphological generator function allows the user to search for lexical entries in the majority language and generate other relevant inflections such as the full paradigm of a verb stem. Other data or technologies can be added as well including audio in the entries or example sentences from an analyzed corpus of text (Arppe et al., 2016).

The second application is a spell-checker integrated into mobile keyboards and operating systems and text editing software. The Divvun and Giellatekno gropus have created a framework to easily create a spell-checker from an existing morphological analyzer and package them into the various applications (Trosterud, 2006).

The third application is an intelligent computer-aided language learning (ICALL) application. The integration of a morphological analyzer allows the learner to practice full paradigms. This application can be paired with a grammar textbook and audio to supplement and enrich the language learner with comprehensible input (Arppe et al., 2016).

Other possible applications are to build syntactic parsers from the analyses given by the morphological analyzer, to create a morphologically tagged corpus, and to create a language model from the corpus of analyzed text by giving weights to the analyzer (Arppe et al., 2016). Another recent development is to create a morphological analyzer by bootstrapping neural models with finite-state morphological generator data (Schwartz et al., 2019).

In the future, I intend to integrate the morphological analyzer for CAY into first and

foremost an intelligent electronic dictionary. I believe having a dictionary that is able to analyze and generate morphologically complex words is beneficial to the CAY community and language learners. Unless you've been trained in CAY grammar, it is difficult to access the full knowledge in a traditional dictionary, such as Jacobson 2012. I am also exited to integrate the analyzer into other language technologies such as spell-checkers, ICALL software, and syntactic/semantic parsers.

3.5 Summary

This chapter introduces the concepts of finite-state morphology including the underlying technology of FSTs and the beneficial properties of bidirectionality, composition, modularity, small size, and quick efficiency. Next, I explained the different FST compilation formalisms, *lexc*, *xfst*, and *twolc*, and the FST tookits, Xerox/PARC, SFST, foma, and HFST. Then I described the other IYU languages with morphological analyzers along with a survey of other languages and their toolkits and formalisms. The last section describes the different uses for a finite-state morphological analyzer including applications such as intelligent dictionaries, spell-checkers, and ICALL applications. The next chapter describes the methodology of creating the CAY finite-state morphological analyzer including the FST toolkit and formalisms.

Chapter 4

PROJECT METHODOLOGY

This chapter explains all of the components I used to create a finite-state morphological analyzer for CAY. The morphological analysis I have implemented is based on the analysis given by Jacobson's grammar book (1995) and dictionary (2012). The underlying form representation emulates Chen's (2018) underlying form representation for Central Siberian Yupik. And I used the Helsinki Finite-State Transducer (HFST) toolkit (Lindén et al., 2011) to compile and compose the six level stack FST morphological analyzer.

I began this project by working through and implementing the grammatical processes from Jacobson's (1995) grammar book chapter by chapter with a test-driven development (TDD) approach using test-by-generation and test-by-analysis. After the necessary initial creation of the FST stack files and scripts to compile and test the resulting FST, I created a test file for each chapter. These test files include individual tests consisting of the underlying form and the surface form of a CAY word from the chapter. After these tests were written, I incrementally implemented the grammatical processes and ran the tests until all (or most of) the tests were passing.

I encountered many different issues and design decisions that needed to be resolved during this incremental TDD. For example, when I was creating the test cases, the main issue came from the many little decisions I had to make concerning the representation of the underlying form. If I decided on a different underlying form representation, I had to go back through the previous test cases and make them consistent. Another design decision came from when I had to decide which FST formalism (Section 3.2) and which FST stack file (Section 4.3) would best handle each grammatical feature. Finally, it was also important to keep the representations of each input and output character(s) straight for each FST stack

layer. Designing and standardizing the input and output of each stack did not allow errors to propagate through the FST stack.

The rest of the chapter is organized by the components of the morphological analyzer. Section 4.1 describes the lexicon, symbols, and special characters necessary to process CAY words. Section 4.2 describes the underlying form representation for CAY words. And finally, Section 4.3 describes the finite-state tools and FST stack structure of the morphological analyzer.

4.1 Lexicon, Symbols, and Special Characters

This section describes the lexicon, morphophonological symbols, and special characters used in the creation of this morphological analyzer. These all come from the analysis from Jacobson's *Yup'ik Eskimo Dictionary* (2012) and *A Practical Grammar of the Central Alaskan Yup'ik Eskimo Language* (1995). Section 4.1.1 gives a list of all of the types of lexical entries. Section 4.1.2 describes the additional morphophonological symbols and special characters that I implemented.

4.1.1 Lexicon

The lexicon that I implemented in the morphological analyzer is based on the analysis of Jacobson's *Yup'ik Eskimo Dictionary* (2012) and *A Practical Grammar of the Central Alaskan Yup'ik Eskimo Language* (1995). These two complimentary resources include a collected lexicon of stems (bases), derivational suffixes (postbases), inflectional suffixes (endings), enclitics, and many other lexical categories. Table 4.1 lists all of the lexical entries that were implemented into the FST *lexc* file (see Section 4.3.2).

I extracted the stems (proper nouns, common nouns, common verbs, and particles) from the bases section in Jacobson's (2012) dictionary. I extracted the derivational suffixes from the postbases section and hand-categorized the derivational suffix types from the lexeme definitions. The inflectional suffixes came from the endings section along with extra refinements in the analysis from the grammar book (Jacobson, 1995). The verb roots, enclitics,

Lexicon Type	**Count**
Nouns	
Proper Noun	342
Common Noun	6920
Dimensional Root	35
Subtotal: Nouns	**7,297**
Verbs	
Common Verb	5341
Emotional Root	59
Postural Root	57
Subtotal: Verbs	**5,457**
Postbases	
Noun-elaborating	105
Verbalizing	112
EtePostbase	42
Verb-elaborating	315
Nominalizing	56
Subtotal: Postbases	**630**
Noun Endings	
ABS	51
REL	57
ABM (and HBC)	51 x 2
LOC	58
TER	58
VIA	58
EQU	51
Subtotal: Noun Endings	**435**

Verb Endings	
IND	88
INT	75
OPT - present	85
OPT - future	82
OPT - negative present	21
OPT - negative future	21
PTP	74
SUB	20
CNN	129 x 9
Subtotal: Verb Endings	**1,627**
Enclitic	21
Particles	258
Ignorative	55
Demonstrative	146
Personal Pronoun	108
Quantitative/Qualitative	90
Numeral	64
Exceptions	82
Total	**16,248**

Table 4.1: Lexicon Count

demonstratives, quantative/qualitative, and numerals came from other lists and charts in the dictionary. The exceptions were created from irregular base+postbase combinations in the postbases section of the dictionary or the grammar book. In total, 16,248 lexical entries were entered into the lexc file.

4.1.2 Morphophonological Symbols and Special Characters

This subsection explains the additional morphophonological symbols not used in the second edition of the dictionary (Jacobson, 2012) and other special characters that were necessary for building the morphophonological analysis. An overview of morphophonological symbols was given in Section 2.4 and the full list of symbols and their functions is given in Appendix A. The first set of additional morphophonological symbols and characters were reintroduced from the first edition of Jacobson's (1984) dictionary. The next set of FST-internal special characters were introduced to indicate phenomena that needed to undergo not only morphophonological rules but also prosodic adjustment rules.

Reintroduced Symbols and Characters

Since I used the lexicon from the second edition of the dictionary (Jacobson, 2012), there were certain morphophonological symbols that were missing from the lexical entries from the prior grammar book by Reed et al. (1977) and the first edition of the dictionary (Jacobson, 1984). The first items that I needed to reintroduce to the lexicon were the five categories of the @ symbol for suffixes attaching to *te*-ending bases (see Table 2.6). Because of the semi-predictable nature of the @ symbol when looking at the first character of the suffix, these categories are not strictly necessary, but there were enough instances where exceptions in the code were needed that implementing the categorization of the @ symbol was well-motivated in terms of simplifying the overall code.

After implementing the five existing categories, I implemented an additional three types of @ symbol categories on certain suffixes not represented in the five existing categories to further simplify the morphophonological code. The first additional category, @^{l}, add an

@ symbol on suffixes starting with *l* for devoicing the *l* (i.e. *l* → *ll*) and dropping the *te* on the base. The next category, @^{y}, came from Jacobson adding an @ symbol on suffixes starting with *y* for the devoicing and affrication processes with *t* (i.e. *ty* → *ts* → *c*). The last additional category, @^{ʕ}, is specially used on the @^{ʕ}*+'(g/t)ur(ar)* 'to continue to V' suffix because it has the unique process of changing *t* → *q*. An example set for all eight categories using the base *ceñirte-* 'to visit' is given in (51).

(51) a. ceñirrniuq[1]
ceñirte @[1]~+ni +'(g/t)uq
ceñirr -ni -uq
visit -A'.to.say -IND.3SG
's/he said s/he is visiting'

b. ceñirrngaituq
ceñirte @[2]~+ngaite° +'(g/t)uq
ceñirr -ngait -uq
visit -FUT.NEG -IND.3SG
's/he won't visit'

c. ceñirutuk
ceñirte @[3]%:(u)te +'(g/t)uk
ceñir -ut -uk
visit -with.another -IND.3DU
'they$_2$ visited each other'

d. ceñireskuni
ceñirte @[4]~—k̲u̲+ni
ceñires -kuni
visit -CNNif.4SG
'if s/he visits'

e. ceñirciiquq
ceñirte @[5]ciiqe +'(g/t)uq
ceñir -ciiq -uq
visit -FUT -IND.3SG
's/he will visit'

f. ceñirrluni
ceñirte @^{l}~+luni
ceñirr *-luni*
visit -SUB.4SG
'and s/he visits'

g. ceñircugtuq
ceñirte @^{y}~+yug +'(g/t)uq
ceñirc *-ug* *-tuq*
visit -DES -IND.3SG
's/he wants to visit'

h. ceñirqurtuq
ceñirte @$^{\varsigma}$+'(g/t)ur(ar) +'(g/t)uq
ceñirq *-ur* *-tuq*
visit -continues -IND.3SG
's/he continues to visit'

From the examples in (51), one can see the extent of variation in one subclass *te*-ending base, Class IVa, with the various @ symbol categories. Using just one stem, *ceñirte-*, one can see that every stem has various alterations. The fricative, *r*, before the *te* has a variation of staying voiceless as in examples (51a, 51b, 51e, 51f, 51g, 51h), becomes voiced as in (51c), or gets *es* added like in (51d). This is just the voiceless fricative plus *te*-ending base subclass. There are also other subclasses to handle such as vowel plus *te*-ending bases (Class IVb), nasal plus *te*-ending bases, short *te*-ending bases, and special *te*-ending bases (Class IVc). Having all the different @ symbol categories explicitly indicated allows for an easier time in creating constraint rules because I can write rules that are sensitive to the specific @ in the string. For example, the special rule only references the @$^{\varsigma}$ symbol in (52) and not the whole *@$^{\varsigma}$+'(g/t)ur(ar)* suffix. This rule states that `t` in the upper-level is realized as `q` in the lower-level when there are any number of other symbols separating the `e` in the *te*-ending and the `@`$^{\varsigma}$ symbol in the upper-level right-hand context.

(52) `t:q <=> _ (') e: (%°:) [Bndry:-AtSign:]* %@`$^{\varsigma}$`: ;`

[1]see Footnote 5 in Section 2.4.2

The second symbol that I needed to reintroduce from the first edition dictionary Jacobson (1984) into the lexicon taken from the second edition dictionary Jacobson (2012) was the ° symbol to mark special *te*-ending bases (Class IVc). The ° symbol is used after special *te* stems and derivational suffixes as an indicator to certain derivational or inflectional suffixes to use a different form or conjugation pattern. Again, this symbol may have been removed from the second edition because it is semi-predictable, with the symbol used on bases that are negative or adjectival in nature. Since this is a semantic feature that isn't represented in a form-only lexicon, it was necessary to add them back into the lexicon. This symbol is especially important for the frequently used subordinative mood inflectional suffixes. As shown in (53), the subordinative mood has two different allomorphs, one regular form, *@^{l}~+lu*, as in (53a), and one irregular form, *@^{5}na*, used only with special *te* bases as in (53b). Without the ° symbol to mark special *te*-ending bases, the analyzer would not know which allomorph to use.

(53) a. niilluni
niite @^{l}~+luni
nii *-lluni*
hear -SUB.4SG
'and s/he heard'

b. assiinani
assiite° @^{5}nani
assii *-nani*
bad -SUB.4SG
'and it was bad'

The third type of special character that was in the first edition dictionary that is not in the second edition was the velar/uvular assimilation special characters, *k̲ g̲*, *g r̲*, *gg r̲r̲*. These characters occur on the first letter of suffixes that are velar or uvular. As previously explained in Section 2.4, they indicate that if the base the suffix is attaching to ends in a velar and the special character is a uvular, then the special character assimilates to become a velar as well, and vice versa with a uvular-ending base and special velar-initial suffix. Jacobson

may have removed these special characters because this phenomenon occurs on the majority of velar/uvular-initial suffixes and it was easier to have it be the default function and just note in the dictionary which suffixes are the exception. But for my purposes, it was easiest to indicate where the assimilation happens rather than finding all of the exceptions to the rule.

(54) a. nerqatartuq
nere –qatar +'(g/t)uq
ner *-qatar* *-tuq*
eat -going.to -IND.3SG
's/he is going to eat'

b. atuqatartuq
atur –qatar +'(g/t)uq
atu *-qatar* *-tuq*
sing -going.to -IND.3SG
's/he is going to sing'

c. ayakatartuq
ayag –qatar +'(g/t)uq
aya *-katar* *-tuq*
go -going.to -IND.3SG
's/he is going to go'

An example of one suffix that utilizes the velar/uvular assimilation special character is the *–qatar* suffix, a uvular-inital suffix, as shown in example (54). The special character assimilation does not apply to vowel ending bases such as (54a) or for bases ending in the same place of articulation such as (54b). It is used in (54c) on the velar-ending base *ayag-*. The *q*-initial suffix becomes a *k*-initial because of the velar ending base to become *-katar* and the *g* gets dropped because of the minus (-) symbol. The resulting word becomes *ayakatartuq*.

In summary, I reintroduced three types of symbols or special characters that were not in the lexicon taken from the second edition dictionary (Jacobson, 2012). The first included the set of eight @ symbol classifications, the second was the ° symbol to mark special *te-* ending bases (Class IVc), and the third was set of special characters to indicate velar/uvular

assimilation.

FST-internal Special Characters

I used FST-internal characters (i.e. characters that don't ever appear in the surface string) to model two processes, following closely the analyses in Jacobson 1995. The first process is known as dropping hatted-*e* and is explained in detail in Section 2.4.4. To illustrate the prosody of words, Jacobson (1995, p.55) used diacritics on the vowels to indicate rhythmic lengthening and rhythmic stress as discussed in Section 2.2.1 and 2.2.2. Rhythmic lengthening is indicated FST-internally by the circumflex diacritic on the vowels (i.e. *â î û ê*), also known as hatted vowels, and rhythmic stress is indicated FST-internally by the acute accent diacritic on the vowels (i.e *á í ú é*). Adding the circumflex and acute accent diacritics to indicate rhythmic lengthening and rhythmic stress is done in the prosodic stress level (see Section 4.3.5) and dropping the hatted-*e* occurs in the prosodic adjustment level (see Section 4.3.6).

The second process and last biggest hurdle to complete in the project was the *ar*-deletion process as explained in Section 2.4.4. In addition to the FST-internal circumflex and acute accent diacritics to indicate prosody, the *ar*-deletion process also required special characters, *a̤ ṳ*, to ensure that the *(ar)*/*(ar*)*/*(ur)*[2] in the the underlying form is still identifiable when it passes through the various FST stack levels. The first step in implementing this process was to replace the *(ar)*/*(ar*)*/*(ur)* to *a̤r*/*a̤r**/*ṳr* in the lexical adjustments level (see 4.3.3) before the *(ar)* passes through the morphophonological and prosodic stress levels (see Section 4.3.4 and 4.3.5), and finally the *ar*-process is handled in the prosodic adjustments level (see 4.3.6).

[2] *(ar)* is used with verbal bases and suffixes, *(ar*)* is used with nominal bases and suffixes because the asterisk is a nominal base specific symbol, and *(ur)*, which only occurs on one specific suffix, follows the same processes as *ar*-dropping.

4.1.3 Summary

This section began with an overview of all of the lexical entries that I extracted from the second edition dictionary. Then I described the symbols I reintroduced from the first edition dictionary to the lexicon which include the set of eight @ symbol categories ($@^1$, $@^2$, $@^3$, $@^4$, $@^5$, $@^l$, $@^y$, $@^ɾ$), the ° symbol to mark special *te*-ending bases (Class IVc), and the set of special characters to indicate velar/uvular assimilation (k̲ g̲, g̲ r̲, gg r̲r̲). Finally, I described the FST-internal special characters which include the set of prosodic diacritics on vowels (â î û ê á í ú é) and special characters, ạ ụ, used for the *ar*-deletion process.

4.2 Underlying Form Representation

This section describes the underlying form representation for CAY words. The main purpose for the representation of the underlying form is to have all the word categories and all the grammatical information in the morphemes be explicitly stated. I took advantage of the similarities in the language of Central Siberian Yupik to CAY and emulated Chen and Schwartz's (2018) grammatical morphotactic representations for the underlying form. My philosophy was that underlying form should output some additional information from the analysis of an inputted word. For example, since particles are uninflected, the underlying form would output the word and an explicit grammatical marker stating that the word is a particle. For example, the particle *cali* would have `cali[Particle]` as the underlying form.

4.2.1 Stems and Derivational Suffixes

The lexical stems (bases) in the dictionary (Jacobson, 2012) are represented in the underlying form. Since the dictionary entries for nouns are in the 'ABS.SG' inflection, I converted them to the base form for the underlying form. The dictionary entries for verbs and particles are used as is for the underlying form. Again, since particles do not have inflection, the underlying form just attaches `[Particle]` to the dictionary entry (i.e. *Particle*`[Particle]`).

Derivational suffixes, known as postbases in the dictionary, are separated with a hyphen

before the dictionary form and the derivational type as shown in Figure 4.1.

$$-Postbase \begin{bmatrix} \text{N} \rightarrow \text{N} \\ \text{N} \rightarrow \text{V} \\ \text{V} \rightarrow \text{V} \\ \text{V} \rightarrow \text{N} \end{bmatrix}$$

Figure 4.1: Derivational Suffix Underlying Form

For example, the past tense derivational suffix would be `--llru[V→V]` in the underlying form where a hyphen is attached to the dictionary entry, which is the minus symbol *llru* and the verb-elaborating derivational type.

4.2.2 Inflectional Suffixes

The noun and verb inflections are where it becomes complicated. Figure 4.2 is an inflectional choices diagram representation of the underlying form for noun inflection. All noun inflections start with a `[N]` indicating the word is a noun. The next choice is one of the CAY noun cases. After that, there are two paths separated by a line where one either chooses the unpossessed noun or possessed noun inflection. The unpossessed noun inflection on the upper level of the figure begins with an `[Unpd]` grammatical marker with another marker choosing number. The possessed noun inflection on the lower level first chooses the person and number along with `[Poss]` indicating possessor. Then one chooses a number with `[Posd]` indicating possessed.

For example, to create the word *angyaq* meaning 'boat', then the base would be `angyar` inflected in the absolutive singular case. To get the underlying form, one would start with `[N]` then choose the absolutive case `[Abs]`. This word is unpossessed singular and so the top path is chosen to get `[Unpd][Sg]`. All together the underlying form is `angyar[N][Abs][Unpd][Sg]`. If a possessed noun is created instead, such as *angyaitnun* meaning 'to their boats', one would start with `[N][Ter]` for the terminalis case, but now choose the lower path. 'Their' would

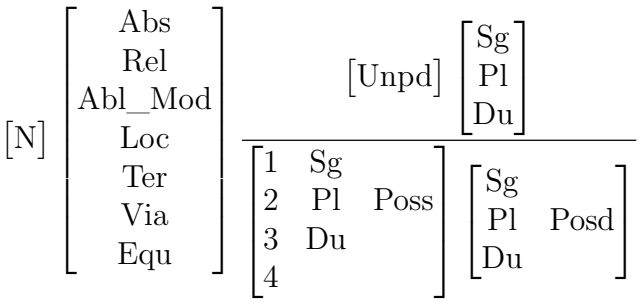

Figure 4.2: Noun Inflection Underlying Form[3]

mean `[3SgPoss]` is chosen and 'boats' is plural so `[PlPosd]` is chosen. All together the underlying form is `angyar[N][Ter][3SgPoss][PlPosd]`.

Figure 4.3 is an inflectional choices diagram representation of the underlying from for verb inflection. All verb inflections start with a `[V]` indicating the word is a verb. Then there is a choice to either take the top path for intransitive inflection or the bottom path for transitive inflection. Both paths then choose one of the CAY verb moods. The optative mood has some extra categories that need to be chosen, either present tense or future tense, plus an optional negative marker. If the top path was chosen for intransitive inflection, then the top path is also chosen for the subject grammatical marker. The subject grammatical marker starts with `S_` and then chooses the person and number. If the bottom path was chosen, then the agent and the patient grammatical markers of the transitive verb are also chosen. The agent grammatical marker begins with `A_` and then chooses the person and number while the patient grammatical marker begins with `P_` and chooses the same.

For example, to create the verb inflection underlying form for the word *qavartuq* meaning 's/he is sleeping', one would use the base `qavar` and start with `[V]` then choose the top path for the intransitive inflection `[Intr]`. The word is in indicative mood so the `[Ind]` marker is chosen and since the top path was used, the top path again is chosen to create `[S_3Sg]` for the subject grammatical marker. All together the underlying form is

[3]This diagram and other underlying form diagrams are just a way of presenting the information for the reader and not an implementation in the analyzer.

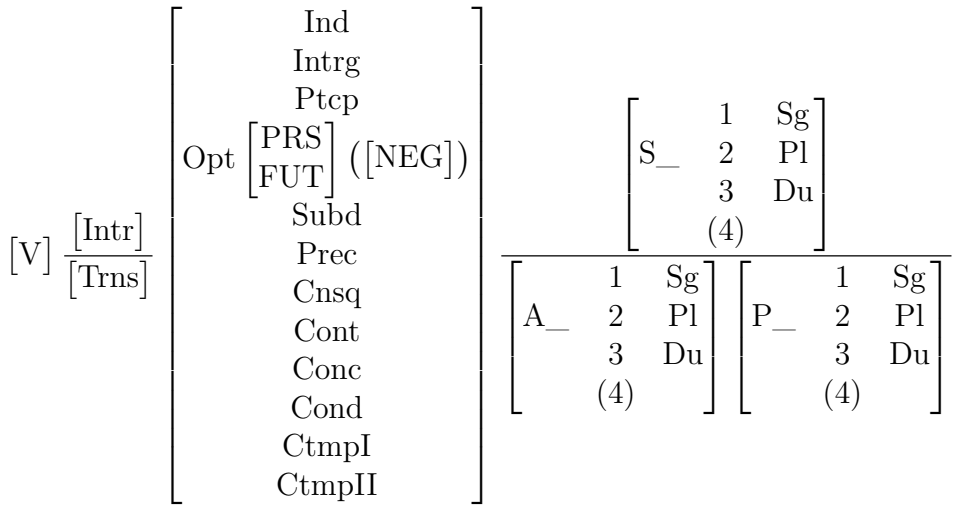

Figure 4.3: Verb Inflection Underlying Form

`qavar[V][Intr][Ind][S_3Sg]`. For a more complicated word such as *nerngamegteki* meaning 'because they themselves are eating them', one would use the base `nere` and start with `[V]` but instead choose the lower path with the transitive inflection `[Trns]`. This word is in the consequential connective mood `[Cnsq]` and since the bottom path was used, the bottom path is followed after mood selection. The agent is 'they themselves' which is the fourth person plural `[A_4Pl]` and the patient is 'them' which is `[P_3Pl]`. All together the underlying form is `nere[V][Trns][Cnsq][A_4Pl][P_3Pl]`.

4.2.3 Other Word Categories

There are other smaller word categories in CAY that require underlying form representations. The first major category is the demonstratives represented in Figure 4.4. Demonstratives begin with a root and either a demonstrative pronoun or demonstrative adverb is chosen. The next grammatical marker is the case marker. The dashed line in the demonstrative cases indicates that some demonstrative adverbs (the bottom path) have an additional case marker for second terminalis. The regular terminalis case has a more definite 'to' meaning while the second terminalis case has a directional 'towards' meaning. After the case marker,

demonstrative pronouns have an additional unpossessed marker while the demonstrative adverbs do not.

$$\textit{Demonstrative}\ \frac{[\text{DemPro}]}{[\text{DemAdv}]}\begin{bmatrix}\text{Abs}\\ \text{Rel}\\ \text{Abl\_Mod}\\ \text{Loc}\\ \text{Ter}\\ \text{Via}\\ \text{Equ}\\ \underline{\text{SecTer}}\end{bmatrix}\ \underline{[\text{Unpd}]\begin{bmatrix}\text{Sg}\\ \text{Pl}\\ \text{Du}\end{bmatrix}}$$

Figure 4.4: Demonstrative Underlying Form

A second word category is for personal pronouns represented in Figure 4.5. Personal pronouns begin with one of two roots, `wang` (for first person) or `ell` (for second or third person) and then include the `[PerPro]` grammatical marker. Then the case and person and number grammatical markers are chosen.

$$\textit{Pronoun}\ [\text{PerPro}]\begin{bmatrix}\text{Abs}\\ \text{Rel}\\ \text{Abl\_Mod}\\ \text{Loc}\\ \text{Ter}\\ \text{Via}\\ \text{Equ}\end{bmatrix}\begin{bmatrix}1 & \text{Sg}\\ 2 & \text{Pl}\\ 3 & \text{Du}\end{bmatrix}$$

Figure 4.5: Personal Pronoun Underlying Form

The third word category is the inflection of quantifier/qualifier construction represented in Figure 4.6.

Other word categories have a grammatical marker attached to the base indicating a special lexical category. Other than the enclitic, these special word categories must take additional noun or verb inflection as indicated in figures above.

$$[\text{Quant\_Qual}] \begin{bmatrix} \text{S\_} & 1 & \text{Sg} \\ \text{A\_} & 2 & \text{Pl} \\ \text{P\_} & 3 & \text{Du} \\ & 4 & \end{bmatrix}$$

Figure 4.6: Quantitive/Qualitive Underlying Form

- $ContentWord$[Ignorative]
- Num[Num] $\begin{bmatrix} \text{Cardinal} \\ \text{Ordinal} \end{bmatrix}$
- $Positional$[Positional]
- = $Enclitic$[Encl]

4.2.4 Summary

This section described the design of the underlying form of CAY words emulated from Chen and Schwartz's (2018) representation of Central Siberian Yupik. It includes descriptions of the underlying form of particles, derivational suffixes, noun and verb inflectional suffixes, and other word categories. Now that the description of morphophonological symbols and underlying form representations are given, the next section will describe the architectural structure of the finite-state morphological analyzer.

4.3 HFST and the FST Stack Structure

This section describes the finite-state tools and FST stack structure of the morphological analyzer. I used the Helsinki Finite-State Transducer (HFST) toolkit (Lindén et al., 2011) because it is open source and because it makes available compilers for all three FST formalisms (*lexc*, *xfst*, and *twolc*) (see Section 3.2). In addition to this, the tools for FST operations necessary in composing and inverting the FSTs are available as well as optimization tools

for creating quicker lookup FSTs. This allows a lot of flexibility in creating morphological analyzers.

Most FST generated grammars would just include two files: the *lexc* file for the lexicon and morphotactics, and either an *xfst* or *twolc* file for the phonological and morphophonological rules. For CAY, I knew I needed a *twolc* file for the morphophonological rules because of the recursive nature of suffix attachment to the base in CAY morphology. The morphophonological rules needed a left-to-right rule application which required observing the surface form on the left-hand side of the morpheme boundary, which have gone through the morphophonological rules, and observing the underlying form for the right-hand side of the morpheme boundary. For this reason, *xfst* replace rules were not the best FST formalism to use. The better FST formalism for this phenomenon is *twolc* parallel two-level constraint rules. This formalism allows one to write rules that use the surface form for the left-hand side of the morpheme boundary and the morphophonological form for the right-hand side of the morpheme boundary through the use of symbol-pairs.

Level	**File Type**	**File Name**	**Phenomena Handled**
1.	`lexc`	*esu.lexc*	lexicon and morphotactics
2.	`xfst`	*esu.lexc.xfst*	lexical adjustments / morphologically conditioned allomorphy / orthography → grapheme
3.	`twolc`	*esu.twol*	morphophonology - phonologically conditioned allomorphy
4.	`twolc`	*esu.stress.twol*	prosodic stress marking on vowels used for prosodic adjustments
5.	`xfst`	*esu.twol.xfst*	prosodic adjustments / grapheme → orthography
6.	`twolc`	*esu.ana.twol*	remove morpheme boundaries

Table 4.2: FST Stack Structure

4.3.1 FST Stack Structure Overview

The FST generated grammar began with the *lexc* file for the lexicon and a *twolc* file for the morphophonological rules. Since I decided to take the orthographically written lexicon directly from the dictionary, I also needed a level between the *lexc* file and the *twolc* file to convert the orthography to unique graphemes (listed in Table 2.2 and 2.1) before being processed by the morphophonological rules. I also needed a level after the *twolc* file to convert the graphemes back into the orthography using the orthographic rules. I decided both of these levels should be in the *xfst* formalism since that will allow the flexibility to add more functionality to those two layers. The second to last layer to be included was a *twolc* file located between the morphophonological *twolc* file and the grapheme-to-orthography file to mark the syllables for prosodic stress needed for prosodic adjustment phenomena. The last layer is just a *twolc* file to remove the morpheme boundary symbol. Table 4.2 shows a summary of each level's file type, file name, and phenomena handled.

Figure 4.7 is an example of the word *aqvatqataraqa* meaning 'I am going to go fetch it' going through all the FST stack levels. Each of the levels will be described in detail in the subsections below.

4.3.2 esu.lexc

The *lexc* file is used to handle the lexicon and morphotactics of the grammar. The analysis of the underlying form will be the upper-side of the FST and the lower-side will be the morphotactic form ready for processing with phonological and morphophonological rules.

The lexicon added to the file was described in detail in Section 4.1. The morphotactics of CAY are described in detail in Section 2.3. The underlying form was described in detail in Section 4.2. These are all the pieces necessary for creating this *lexc* file. An excerpt of the lexc file showing all of the pieces and how they fit together is shown in Appendix B. A diagram showing a high level view of the finite-state morphotactic diagram is given in Figure 4.8.

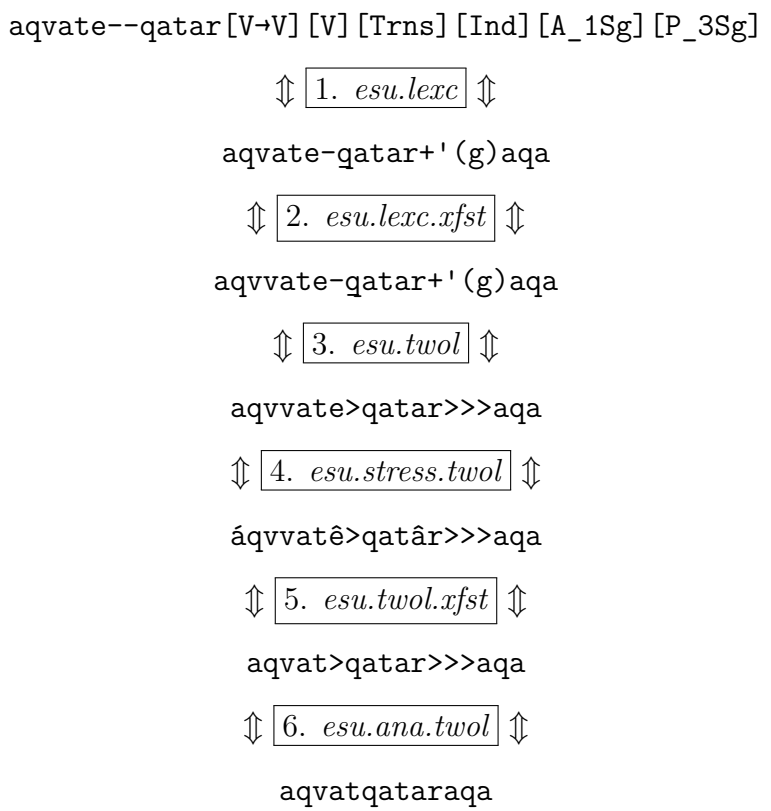

Figure 4.7: Example Word Passing Through FST Stack

The *lexc* file begins with the keyword `Multichar_Symbols`, and the white-space separated symbols listed after it are all the symbols in the *lexc* file that should be handled as one arc in the FST. I've included all of the graphemes (`vv`, `ll`, `ss`, etc.) and grammatical markers (`[N]`, `[V]`, `[V→V]`, `[Ind]`, etc.) since I handle them FST-internally as one arc label. After that, the file must begin with a lexicon category called `Root` using the `LEXICON` keyword. This tells the program where to begin the morphotactics. My `Root` lexicon has pointers to other lexicon categories such as `Particle`, `NounBase`, and `VerbBase`. The lexemes inside the lexicon categories either have the lexeme as is or one can use a colon to specify the upper-side and the lower-side of the FST. For example, the postbase, `--cuar(ar*)[N→N]:-cuar(ar*)`, has both a hyphen and the `[N→N]` grammatical marker in the upper-side underlying form and the lower-side does not include them in the morphotactic form. To end the word, a

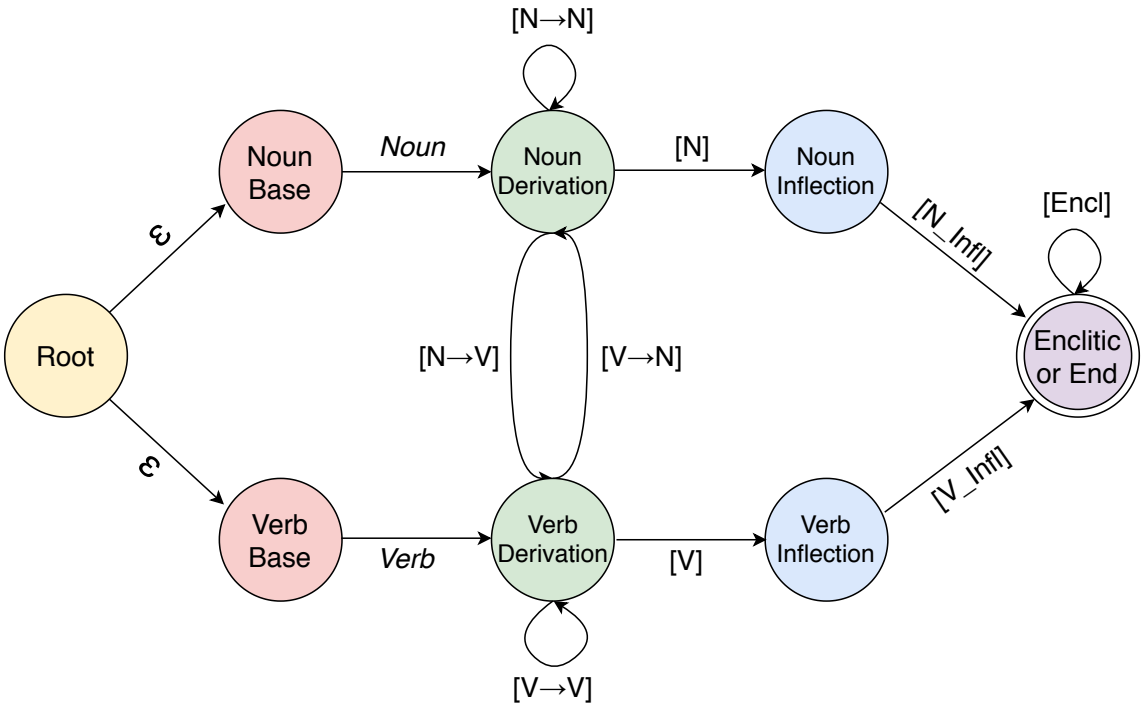

Figure 4.8: Yup'ik Morphotactics Diagram

special lexicon called `#` indicates the end of the string of symbols.

When the *lexc* file is compiled, the resulting FST will have the underlying form as input and output the morphotactic form of the word. Examples in (55) show the representation of individual aligned lexemes. These examples show the underlying and morphotactic forms from verb, noun, and particle words.

(55) a. Underlying Form: `aqvate --qatar[V→V] [V][Trns][Ind][A_1Sg][P_3Sg]`
Morphotactic Form: `aqvate -qatar +'(g)aqa`

b. Underlying Form: `arnar --cuar(ar*)[N→N] [N][Abl_Mod][Unpd][Pl]`
Morphotactic Form: `arnar -cuar(ar*) %~nek`

c. Underlying Form: `cali[Particle] =llu[Encl]`
Morphotactic Form: `cali =llu`

4.3.3 esu.lexc.xfst

This *xfst* file sits between the *lexc* and morphophonology *twolc* file. This file will do any preparation needed before the morphophonology level. In this level, the processes include lowercasing of any proper nouns,[4] converting the orthographic rules in the lexemes to graphemes, replacing single symbols into multiple processable symbols, and deciding which suppletive allomorph to use based on the base's word class.

(56) a. `"A" -> a, "C" -> c, "E" -> e ;`
b. `v -> vv || [ Stop | VlessFric ] _ , _ Stop ;`
c. `"(ar)" -> "ạ" r ;`

Examples of the first three processes are described in (56). The first process, shown in (56a), has any lowercasing of capital symbols as shown in proper noun lexemes. Only the first three letters in the CAY alphabet are shown. The second process, shown in (56b), converts the automatic devoicing of the *v* from the orthographic rules to the grapheme `vv` in the environment where there's either a stop or voiceless fricative on the left-hand side or a stop on the right-hand side. The third process, shown in (56c), has to do with *ar*-deletion explained in Section 4.1.2.

The fourth process deals with suppletive allomorphs. For the majority of the derivational suffixes, the morphophonological symbols attached to the suffix definition are sufficient to define the phonological rules for all word classes (phonological allomorph), but there exists a small set of derivational suffixes where the allomorph is not phonologically predictable for every word class (suppletive allomorph). The suppletive allomorphs are chosen by replace rules in this level. For example, CAY's compound verbal suffix meaning 'to let, allow, permit, cause, or compel one to V' is `-@`^{5}`+cete/.vkar[V→V]` where `@`^{5}`+cete` is used with consonant-ending bases and `.vkar` is used with vowel-ending bases, while *te*-ending bases

[4]Lowercasing allowed me to simplify my implementation by not needing to handle uppercase letters in the FST stack. Another advantage was that the FST did not produce an overgeneration of redundant capitalized forms in my test suite. A disadvantage was that this required me to preprocess my test data for analysis.

can use either allomorph. The way this is handled is to first add `@5+cete/.vkar` as one of the `Multichar_Symbols` in the *lexc* file so it is treated as a single symbol. Then in this *xfst* file there are three rules that replace that single symbol into the various allomorphs as shown in (57).

(57)
```
1.  "@5+cete/.vkar"  ->  "@5" "+" c e t e || Con _       .o.
2.  "@5+cete/.vkar" (->) "@5" "+" c e t e || t e ("°") _ .o.
3.  "@5+cete/.vkar"  ->  "." v k a r                      ;
```

The first replace rule in (57) replaces the single symbol, as defined by the double quotes around the characters, into the `@5+cete` sequence of symbols, as defined by the spaces in between each symbol, in the environment of a consonant on the left-hand side of the symbol. The second rule optionally replaces the symbol into the same `@5+cete` sequence in the environment of a *te*-ending base on the left-hand side. Notice that the optional replace rule `(->)` creates a branching decision and allows *te*-ending bases to not apply this rule as well. The third rule is the default case where any remaining symbols are replaced into the `.vkar` sequence of symbols. This is necessary because the program, not knowing whether the set of vowels and consonants in the language has complete representation over the data, will handle the case where a multi-character symbol has appeared outside of those previously defined environments.

When the *xfst* file is compiled, the resulting FST will have the morphotactic form as input and output the modified morphotactic form of the word. An excerpt of the *esu.lexc.xfst* file is in Appendix C. The examples in (58) show the representation of individual lexemes aligned in the *xfst* file. These examples show the morphotactic and modified morphotactic forms showcasing the different processes used in this level.

(58)
```
a.          Morphotactic Form:  aqvate   -qatar  +'(g)aqa
   Modified Morphotactic Form:  aqvvate  -qatar  +'(g)aqa

b.          Morphotactic Form:  arnar  -cuar(ar*)  %~nek
   Modified Morphotactic Form:  arnar  -cuaṛar*    %~nek
```

c.
Morphotactic Form:	`cali`	`@`^{5}`+cete/.vkar`	`@`^{l}`~+luku`
Modified Morphotactic Form:	`cali`	`.vvkar`	`@`^{l}`~+luku`

4.3.4 esu.twol

This morphophonological *twolc* file is the main powerhouse behind the morphological analyzer. I implemented most morphophonological rules as two-level constraints on symbol-pairs. The two-level constraint rules define left-hand and right-hand context constraints on a pairs of symbols where the upper-level symbol is mapped to the lower-side symbol. These *twolc* constraint rules are applied in parallel rather than the ordered fashion of the replace rules in *xfst* files. The benefit of this formalism is that these rules can define the left-hand or right-hand context constraints on just the upper-level symbol, just the lower-level symbol, or both in a symbol pair. The benefit is that the left-hand context can define constraints on the lower-level and the right-hand context can define constraints on the upper-level to model CAY's recursive rule application.

Because two-level rules represent an equal-length string relation, each symbol in the upper-level is paired with another symbol in the lower-level. Counting the zeros as a symbol that will be deleted, each string will have the exactly the same length (Beesley and Karttunen, 2003b). In this *twolc* file, the upper-level represents the morphotactic form and the lower-level represents the surface form (or similar to the surface form). The set of all possible symbol-pairs in the FST either comes from the symbols in the `Alphabet` section or from the symbol-pair constraint rules.

My design choice was to have a CAY grapheme map to the same grapheme, and all morphophonological symbols map to a generic morpheme boundary symbol `>`. This way the morpheme boundaries will be preserved for analysis and can easily be removed in later levels. All of the one-to-one grapheme pairs and morphological symbol-to-morpheme boundary pairs for CAY are defined in the `Alphabet` section. Every grapheme or symbol that is mapped to a different symbol in the morphophonological processes described in Section 2.4 gets its own *twolc* rule. In (59), an example word using the morphological rule that drops base final

consonants, `-`, is symbol pair aligned.

(59)
```
Modified Morphotactic Form:  a  ng  y  a  r  -  ll  e  r
      Modified Surface Form:  a  ng  y  a  0  >  ll  e  q
```

For the word in (59), the base final `r` is dropped since the morpheme boundary has the consonant dropping morphophonological symbol. Dropping symbols is indicated by a zero in the lower-level. The twolc rule for dropping the `r` is defined in (60a).

(60) a.
```
"r:0 = Rule for Minus"
r:0 <=> _ Bndry:* %-: ;
```

b.
```
"r:q = word#"
r:q <=> _ (%*:) [ .#. | %= ] ; ! Word Final
```

The *twolc* rule begins with a one-line description of the rule in double quotes. The next line specifies the symbol-pair constraint rule specifying that the symbol-pair, `r:0` indicating that the `r` is dropped, is constrained on the right hand context where any number of morphophonological symbols occur between the symbol-pair and the minus symbol. The next rule, (60b), just specifies `r:q` when an optional asterisk separates the end of the word or an enclitic symbol. This next example in (61) is a bit more complicated but shows the power of having two-level constraint rules.

(61)
```
Modified Morphotactic:  a  r  n  a  r  :  ~  (ng)  u  +  '  (g/t)  u  q
      Modified Surface:  a  r  n  a  0  >  >   >   u  >  >    g    u  q
```

For the word *arnauguq* meaning 'she is a woman', the morphophonological rules have `r`-dropping from the colon symbol, and adds `(g/t):g` when the base ends in two vowels. The colon symbol means a velar or uvular consonant is dropped when there are single vowels on both sides of the consonant. In this case, the uvular `r` consonant has a single `a` on the left-hand context and a single `u` on the right-hand context. The resulting base is `arnau-`. The next suffix sees it is a base ending in two vowels and adds the `g` to get *arnauguq*. Notice that one vowel comes from the first morpheme and the other vowel comes from the second morpheme.

In previous attempts to model this phenomenon of left-to-right rule application with ordered replace rules, this example word, *arnauguq*, would not work correctly because there is no universal top-down "order of operations" with CAY morphophonological symbols in Jacobson's (1995) analysis. The whole set of morphophonological rules has to run left-to-right.

To model the left-to-right rule application of CAY, the technique is to create *twolc* rules that reference only the lower-level on the left-hand context and only the upper-level on the right-hand context. This would indicate that the "surface" form (already morpheme attached base form) to the left is attaching to the suffix (morphotactic form) to the right. The two two-level constraint rules in (62) implement this morphophonological phenomenon.

(62) a.
```
"r:0 = Rule for Colon" ! ar(*):(~)(ng)a -> ii
r:0 <=> [:Con|.#.] :FVow _
        (%*:) %:: (%~:) %(ng%): FVow: [Bndry:|Con:|.#.];
```

b.
```
"(g/t):g = VVFinalBase"
%(g%/t%):g <=> :Vow [:%>|:0]* :Vow [:%>|:0]* _ ;
```

The first rule in (61) is the `r`-dropping from the colon symbol. The rule states that the `r` drops when there is a consonant preceding a single full vowel in the surface form on the left-hand context, and if the specific sequence of morphophonological symbols precedes a suffix that begins with a single full vowel in the morphotactic form on the right-hand context. The second rule for `(g/t):g` relies entirely on the left-hand context lower-level surface form. It searches for two vowels each separated by any number of dropping or morpheme boundary symbols.

When the *twolc* file is compiled, the resulting FST will have the modified morphotactic form as input and output the modified surface form of the word. An excerpt of the *esu.twol* file is in Appendix D. The examples in (63) show the aligned symbol-pairs in the lower and upper-levels for a variety of words showcasing the different morphophonological phenomenon used in this level of the FST stack.

(63) a.
Modified Morphotactic Form: k a n a q ll a g % : (e) t
Modified Surface Form: k a n a q ll i 0 > > i t

b.
Modified Morphotactic: a k u t a r - - l i - l a r : ~ (ng) ạ + m i
Modified Surface: a k u c 0 0 > > 0 i > l a 0 > > > ạ > m i

c.
Modified Morphotactic Form: e l i t e @^{y} ~ + y u g @^{l} ~ + l u n i
Modified Surface Form: e l i c 0 > > > 0 u g > > > l u n i

4.3.5 esu.stress.twol

This *twolc* file handles the phenomena of adding prosodic stress diacritics as described by Jacobson's grammar book (1995, pg 55). These diacritics will be referenced in the next level in morphological functions such as hatted-*e* dropping, *ar*-deletion, and dropping for specific suffixes. Prosodic stress in CAY describes many different processes such as rhythmic lengthening, automatic gemination, and rhythmic stress.

The *twolc* formalism works nicely with the five-rule analysis for prosodic stress defined in Jacobson's grammar book (1995, pg 55). Each rule is implemented with one two-level constraint rule for an unstressed-stressed symbol-pair where the right-hand and left-hand contexts only constrain the lower-level syllables. The first rule in the analysis is for rhythmic lengthening of vowels as defined in (64). This rule states, "in a sequence of simple open syllables, [of the form CV and also just V if at the beginning of the word], every second one gets rhythmic length unless it is at the end of the word" (Ibid., pg 55).

(64)
```
! CVCV~[V|CC|C'] ! Rule 1
Vx:Vy <=> [.#.|:Con] B:* :Vow B:* :Con B:* _ ~[ B:* :Vow :* | B:*
          :Con B:* :Con :* | B:* :Con B:* :' :* ] .#. ;
    except
        _ B:* (:Vow) B:* (:Con) B:* [.#.|:%=] ; ! final syllable
    where Vx in Vow
          Vy in VowHatted
    matched;
```

The two-level constraint rule in (64) uses unstressed and hatted (rhythmic lengthened) vowel symbol-pairs and constrains them on the left-hand context to the second simple open syllable (i.e. CVCV) on the lower-level so that the first syllable's vowel stays unstressed in the lower-level. If the first syllable's vowel was stressed or hatted then this constraint rule would not apply. On the right-hand context it is constrained to only simple open syllables, so it negates the contexts where there is either another vowel (not simple), or double consonants to close the syllable. The except keyword also constrains the rule by blocking the rule in the context of the final syllable in the word. The example in (65) shows a word having a series of simple open syllables. The 2nd, 4th, and 6th syllables use the constraint rule and the vowels become hatted indicating rhythmic lengthening. Notice the 8th syllable which is up for rhythmic lengthening but does not use the rule because it is the last syllable in the word.

(65)
Modified Surface Form: `na llu ya gu c>>>a qu na ku`
Stress Surface Form: `na llû ya gû c>>>a qû na ku`

The rest of the constraint rules work similarly but instead of hatting, they add an acute accent diacritic to the vowel to indicate stress. When the *twolc* file is compiled, the resulting FST will have the modified surface form as input and output the stress surface form of the word. An excerpt of the *esu.stress.twol* file is in Appendix E. Examples in (66) show the aligned syllables in the lower and upper-levels for a variety of words showcasing the different prosodic stress phenomena used in this level of the FST stack.

(66) a.
Modified Surface Form: `iq vva>ll rru>n ri t>>>>uq`
Stress Surface Form: `íq vva>ll rrú>n ri t>>>>uq`

b.
Modified Surface Form: `qa ya>rr pa >>li >>>yu>ll rru>>>u >nga`
Stress Surface Form: `qa yá>rr pa >>lî >>>yú>ll rru>>>ú >nga`

c.
Modified Surface Form: `i kam ra>rr pa >>li>n ri t>>>>uq`
Stress Surface Form: `i kám rá>rr pa >>lí>n ri t>>>>uq`

4.3.6 esu.twol.xfst

This *xfst* file implements the rest of the morphophonological, phonological, and prosodic stress adjustment processes that were not handled by the morphophonological *twolc* file to output the orthographic surface form, so this file contains a lot of rules. It begins by rerunning the orthographic automatic devoicing rule used in *esu.lexc.xfst* because the *twolc* rules created further environments for it. It drops hatted-*e* vowels (67a). It inserts *e*-vowels into triple consonant clusters (67b). It drops middle vowels in triple vowel clusters. It deals with any replacements in special suffix patterns (67c). It removes unnecessary apostrophes from automatic gemination. It adds apostrophes to stop automatic devoicing in consonant clusters. And finally it removes stress diacritics from the vowels and changes graphemes to the orthography.

(67) a.
```
ê -> 0 .o. [..] -> e || p _ ">"* p , t _ ">"* t , ...
```
b.
```
[..] (->) e || Con ">"* Con _ ">"* Con .o.
[..] -> e || Con _ ">"* Con ">"* Con ;
```
c.
```
t (->) c || #. _ [e|é] ">"* t ;
```

The process of *ar*-deletion (68) is also handled here using the stress diacritics from the previous level. It required nine different ordered replace rules composed together with the `.o.` operator to implement. Rules (68.1) and (68.5) implement the rule that hatted vowels keep their lengthening by doubling and adding an apostrophe in the middle of the vowels to negate any automatic gemination, and then drop the `ar` in those instances with (68.2) and (68.6). Rules (68.3) and (68.4) just drop `ar` when the previous syllable has double vowels, or when the previous syllable is stressed.

The last three *ar*-deletion rules define the process where the stress is on `ar` that is getting deleted (68.8), so the stress has to shift to the previous syllable (68.7). This causes rhythmic stress to be irregular and so an apostrophe is added to indicate the explicit stressed syllable. But this explicit apostrophe is unnecessary for native speakers since they will automatically know the stress shifts and so it becomes optional, indicated with the optional replace operator

`(->)` (68.9).

(68) **`define arDeletion`**

1. `"â" -> a "'" a,`
   `"î" -> i "'" i,`
   `"û" -> u "'" u || _ ">"* Con ">"* "ạ" q ">"* .#. .o.`
2. `"ạ" q -> 0 || _ ">"* .#. .o.`
3. `["ạ"|"ą́"] [r|rr] -> 0 || Vow ">"* VowStress ">"* Con ">"* _ ">"* Con .o.`
4. `"ạ" [r|rr] -> 0 || Con ">"* VowStress ">"* Con ">"* _ ">"* Con .o.`
5. `"â" -> a "'" á,`
   `"î" -> i "'" í,`
   `"û" -> u "'" ú || _ ">"* Con ">"* ["ạ"|"ą́"] [r|rr] ">"* Con .o.`
6. `["ạ"|"ą́"] [r|rr] -> 0 || Vow "'" VowStress ">"* Con ">"* _ ">"* Con .o.`
7. `a -> á "'",`
   `e -> é "'",`
   `i -> í "'",`
   `u -> ú "'" || _ ">"* Con ">"* "ą́" [r|rr] ">"* Con .o.`
8. `"ą́" [r|rr] -> 0 || VowStress "'" ">"* Con ">"* _ ">"* Con .o.`
9. `"'" (->) 0 || VowStress _ ">"* Con ;`

This process of adding the apostrophe to explicitly stress the syllable is shown in example (69).

(69)

Stress Surface Form:	`igár`	`>>>turạ́r`	`>>tut`
Separated Surface Form:	`igar`	`>>>tu'r`	`>>tut`

When the final *xfst* file is compiled, the resulting FST will have the stress surface form as input and output the surface form of the word with morpheme boundaries. The final step is to remove the morpheme boundary symbols that were originally all morphophonological symbols in the morphotactic form. An excerpt of the *esu.twol.xfst* file is in Appendix F. Examples in (70) show the morpheme separation in the lower and upper-levels for a variety of words showcasing the different morphophonological phenomenon used in this level of the FST stack.

(70) a. Stress Surface Form: `áqvvatê >qatâr >>>aqa`
Separated Surface Form: `aqvat >qatar >>>aqa`

b. Stress Surface Form: `míngq >>>sug >>tut`
Separated Surface Form: `mingeq >>>sug >>tut`

c. Stress Surface Form: `angûte >>>t`
Separated Surface Form: `anguce >>>t`

d. Stress Surface Form: `árná >cuárạr >>>nek`
Separated Surface Form: `arna >cuar >>>nek`

4.3.7 esu.ana.twol

The final *twolc* file simply removes the morpheme boundary symbols and converts the enclitic symbol to an orthographic hyphen. This is done with simple *twolc* rules given in (71).

(71) a.
```
"Remove morpheme boundary"
%>:0 <=> _ ;
```

b.
```
"Convert enclitic to hyphen"
%=:%- <=> _ ;
```

Both rules in (71) are simple replace rules that could have also been compiled as regular expression replace rules. The final output of this composed FST stack is the orthographic surface form of CAY words. Examples in (72) show the removal of the morpheme boundary symbols and conversion of the enclitic to a hyphen.

(72) a. Separated Surface Form: `aqvat>qatar>>>aqa`
Surface Form: `aqvatqataraqa`

b. Separated Surface Form: `arna>cuar>>>nek=llu`
Surface Form: `arnacuarnek-llu`

4.4 Summary

This chapter has presented an overview of the components necessary to create a finite-state morphological analyzer for CAY. Using an incremental test-driven development (TDD) approach, I created this six level FST morphological analyzer for CAY using the analysis given

by Jacobson's (1995, 2012) grammar book and dictionary, emulating Chen and Schwartz's (2018) underlying form representation, and compiled using HFST (Lindén et al., 2011). In the next chapter, I will describe how I created the test suite for TDD and a corpus test for evaluating the finite-state morphological analyzer.

Chapter 5

EVALUATION METHODOLOGY

In this chapter I will first describe the test suite I created for test-driven development (TDD) in Section 5.1 and then the corpus test I used for evaluating the finite-state morphological analyzer in Section 5.2.

5.1 Test Suite

In this section I'll describe the test suite created for test-driven development (TDD) of the finite-state morphological analyzer. TDD is a software development process where writing test cases, the individual tests, drives the development of the code. The test suite is the collection of test cases. The TDD process is shown in Figure 5.1.

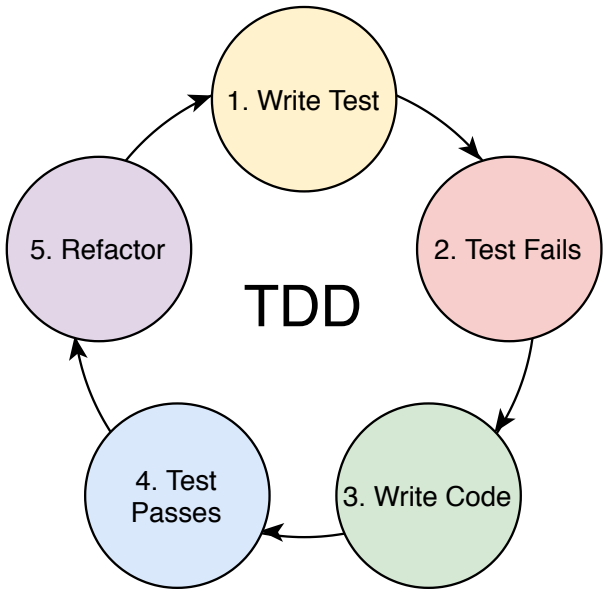

Figure 5.1: Test-Driven Development Cycle

The first step in TDD is writing the tests. The second step is to run the test and make sure the test is failing. If the test is passing then one goes back to the first step and write another test. The third step is to write as much code to get the test passing in step four. The fifth step is to refactor the code which may include removing duplicate code, renaming variables and functions, and splitting large functions for readability. In the fourth and fifth step, a major requirement includes regression testing, re-running all previous tests to make sure the new code does not break previous tests or introduces new bugs.

I began this project by working through and implementing the grammatical processes from Jacobson's (1995) grammar book chapter by chapter with a TDD approach. After the necessary initial creation of the FST stack files and scripts to compile and test the resulting FST, I created a test file for each chapter. These test files include individual test cases consisting of the underlying form and the surface form of a CAY word from the chapter. After these tests were written, I incrementally implemented the grammatical processes and ran the tests until all (or most of) the tests were passing.

5.1.1 Test Cases

The test cases I have written come from Jacobson's (1995)) grammar book. Each test case consists of an underlying form and the surface form of a word given in the grammar description examples. A variety of test cases are shown in (73).

(73) a. `cali--nrite[V→V][V][Intr][Ind][S_3Sg]  calinrituq`
b. `aana[N][Abs][1SgPoss][SgPosd]  aanaka`
c. `atkug-:~(ng)u[N→V][V][Intr][Ind][S_3Sg]  atkuuguq`
d. `ataam[Particle]=qaa[Encl]  ataam-qaa`
e. `wang[PerPro][Abs][1Sg]  wiinga`
f. `tau[DemPro][Abs][Unpd][Sg]  tauna`

The main reasoning for creating test cases before writing code was that there would be examples of the underlying form representation before writing any code. These many little

decisions helped in the design of the lexical entries which was the first step in developing the morphological analyzer.

I created a total of 2,792 test cases from 24 chapters of Jacobson's (1995) grammar book. These test cases are run using test-by-generation and test-by-analysis. Generation means the FST takes the underlying form as input and outputs the surface form, while analysis means the FST takes the surface form as input and outputs the underlying form. Then the testing script takes output of the test and compares it to the correct answer and classifies in four ways as shown in Table 5.1.

Key	**Description**
NO	No output
OI	Only incorrect output
UC	Unambiguous, correct output
AC	Ambiguous, correct output

Table 5.1: Test Suite Output Classification

Key NO of the test suite output given in Table 5.1 indicates the FST did not output anything, which means it did not parse the input. There could be many reasons for no output including a typo in the input, an incorrect morphotactic sequence, r the system just didn't have that morphotactic sequence in the *lexc* file yet. Key OI indicates the FST outputs something but the expected answer, the gold answer, is not in the output. Key UC indicates there is one output item and it is the gold answer. This is the most desirable classification because there is no ambiguity in the output. Key AC indicates there are multiple output items and the gold answer is in the output. This ambiguity in output may come from many reasons including lexical ambiguity, variation in spelling, and morphophonological variation.

5.1.2 Test Analysis

Table 5.2 provides the type and token counts for the underlying form lexical entries in the 2792 test cases in the test suite. This table is similar to the Table 4.1 in Section 4.1.1. There

Lexicon Type	Types	Tokens
Nouns		
Proper Noun	5	8
Common Noun	209	868
Dimensional Root	9	42
Subtotal: Nouns	**223**	**918**
Verbs		
Common Verb	244	1,555
Emotional Root	7	25
Postural Root	12	26
Subtotal: Verbs	**263**	**1,606**
Postbases		
Noun-elaborating	27	177
Verbalizing	27	298
EtePostbase	5	22
Verb-elaborating	92	932
Nominalizing	21	171
Subtotal: Postbases	**172**	**1,600**
Noun Endings		
ABS	26	543
REL	22	69
ABM (and HBC)	11	166
LOC	5	107
TER	9	55
VIA	10	31
EQU	7	17
Subtotal: Noun Endings	**90**	**834**

Verb Endings		
IND	41	1,130
INT	24	102
OPT - present	27	129
OPT - future	8	20
OPT - present negative	2	2
OPT - future negative	7	12
PTP	13	37
SUB	14	156
CNN		
CNNbc	10	49
CNNbf	26	41
CNNif	10	23
CNNth	9	19
CNNwl	9	19
CNNwn	13	10
CNNwv	5	7
Subtotal: Verb Endings	**218**	**1,756**
Enclitic	8	44
Particles	5	5
Ignorative	7	29
Demonstrative		
DemPronoun	59	79
DemAdverb	30	75
Personal Pronoun	2	23
Quantitative/Qualitative	13	32
Numeral	13	43
Total	**1,103**	**7,044**

Table 5.2: Test Suite Types and Tokens

is a large difference in comparing the total lexicon count with the type count in the respective sections. The type count is nowhere near comprehensive of the lexicon but hopefully the words chosen in the grammar give a representative sample of the grammatical processes.

5.2 Corpus Test

In this section, I'll describe the corpus test I used for evaluating the finite-state morphological analyzer. This test consists of two different sources of Yup'ik words, the Yup'ik Bible (American Bible Society, 2014) and a collection of culture books. The Yup'ik words were then

preprocessed for normalization before they were analyzed by the finite-state morphological analyzer.

5.2.1 Data Sources

The corpus of Yup'ik words come from two different sources, the Yup'ik Bible (American Bible Society, 2014) and a collection of culture books. The Yup'ik Bible, called *Tanqilriit Igat*, is a translated work by native Yup'ik speakers from the Bethel area and the American Bible Society (Enoch, 2015). Because this work is a translation, it is processed separately from the other culture book sources.

The second source of Yup'ik words is a collection of culture books. These bilingual Yup'ik/English books have transcribed interviews and stories from knowledgeable elders from many Yup'ik speaking communities. These include communities that speak in different dialects. This collection includes 8 different books done by Ann Fienup-Riordan, Alice Rearden, and Marie Meade (John et al., 2003; Fienup-Riordan and Rearden, 2011, 2013, 2014, 2016a,b; Fienup-Riordan et al., 2017, 2018), and another book of compiled stories published by the Lower Kuskokwim School District (Tennant and Bitar, 2000). These works are considered as a separate source from the Yup'ik Bible data because the texts are originally Yup'ik and it's the English that is translated.

5.2.2 Data Preprocessing

Before the Yup'ik words were used as input for the morphological analyzer, I had to write a script to preprocess the words. This preprocessing script takes a large text file of words and splits them by whitespace. Then everything is lowercased and all punctuation and numbers are removed except hyphens, apostrophes, and ligature marks (u͡g, *m̄*, *n̄*, *n̄g*). Word internal and ending apostrophes are kept and converted to a special modifier letter apostrophe (U+02BC). The last thing the script does is remove explicitly English words, which means removing words with letters not in the Yup'ik alphabet (b, d, f, h, j, o, x, z) and disallowed

sequences of letters such as *ee*. It does not remove English words that use only CAY letters, like 'cat'.

5.2.3 Data Output

After creating the word list out of the test corpus, I used the finite-state morphological analyzer to process all of the words and output all possible parses it finds (i.e. all possible underlying forms for those surface forms, according to the implemented analyses). I don't have gold (expected) results for these parses and so the coverage metric consists of whether the word parses or not. Then I analyzed the output of all parses for any possible correlations in the results.

Additionally, I hand classified the output of a sampling of words from the corpus. The technique I used to sample the words in the corpus was to first split the corpus into "buckets". The first split was for words that parsed and words that did not parse. The words that did not parse became one bucket. For the words that parsed, I first sorted the words by the number of outputs. Then I split the list into four equal-sized groups with the upper quartile bucket of words having a large number of possible parses and the lower quartile bucket having very few possible parses. Then from each of the five buckets, I took a random sample of 30 words and hand classified each word. I hand classified the output for both the culture books and the Yup'ik Bible corpora for a total of 300 words (30 words * 5 buckets * 2 corpora). The parsed word output classification follows the same classification as the test suite in Table 5.1 but two additional classifications were necessary for hand classifying the analyzer output as shown in 5.3.

The first additional classification is the unknown classification. The unknown classification is used with words that either I don't know the semantic sense of the word and thus cannot choose the correct classification or the morphophonology in a word class is used differently and the analysis in the dictionary and grammar may be incomplete. Many of the disputed words had either a change from consonant dropping/retaining or *e*-dropping/retaining that differed from the analysis in the dictionary and grammar.

Key	Description
UN	Unknown
NO	No analysis
OI	Only incorrect analysis
OG	Overgeneration
UC	Unambiguous, correct analysis
AC	Ambiguous, correct analysis

Table 5.3: Corpus Test Output Classification

The second additional classification is used for overgeneration. These words are analyzed to have possible parses but are misspelled either by orthographic rules or OCR errors and thus should not have parsed.

The words that did not parse are classified according to Table 5.4.

Error Key	Description
X.PRG	Program Error
X.PRG.LEX	- Missing Lexical Entry
X.PRG.TAC	- Morphotactics
X.PRG.PHO	- Morphophonology
X.MIS	Misspelled
X.MIS.ORT	- Orthography
X.MIS.OCR	- OCR
X.MIS.PRE	- Preprocessing
X.CMX	Code-mixing
X.CMX.ENG	- English

Table 5.4: Incorrect Parse Error Classification

The first classification is an error from the morphological analyzer. This may be the result of missing lexical entries, missing morphological processes, or incorrectly implemented morphological rules. The second classification is from misspelled words. These may be the result of not following orthographic rules, OCR errors, the preprocessor introduced an error. The third classification is from code-mixing. Many English words may act as noun or verb bases and a special postbase can be use to attach other derivational or inflectional suffixes.

The morphological analyzer does not handle this code-mixing and will not parse the word.

5.2.4 Data Analysis

Table 5.5 provides the type and token counts for the words in the culture books, Yup'ik Bible, and both sources combined. One interesting feature in the data is that not many of the types appeared in both data sources. Only 11K types out of the approximately 100K types in both sources overlapped. This may have to do with the large number of word inflection and derivation generating unique words from a single base. But since the two sources are different genres, the set of lexical bases they use may be different which would result in unique words between the culture books and Yup'ik Bible corpora.

	Types	**Tokens**
Culture Books	108,326	307,356
Yup'ik Bible	98,672	324,720
Culture+Bible	195,710	632,076

Table 5.5: Types and Tokens

Figure 5.2 shows a common representation called the rank-frequency distribution relating to Zipf's law. The graph shows the ranking of the word in a sorted frequency list by the word frequency. The most frequent word in the corpus *tua-i*, meaning 'and', accounts for nearly 3% of words in the corpus. The second most frequent word *tauna*, meaning 'that', accounts for 0.8% of words in the corpus by comparison.

5.3 Summary

This chapter has presented an overview and an analysis of the two types of evaluations used in this project. The first evaluation method was a test suite for TDD of the morphological analyzer to evaluate the coverage of grammatical phenomena described in the grammar book. The second method was a corpus test from two different sources, culture books and

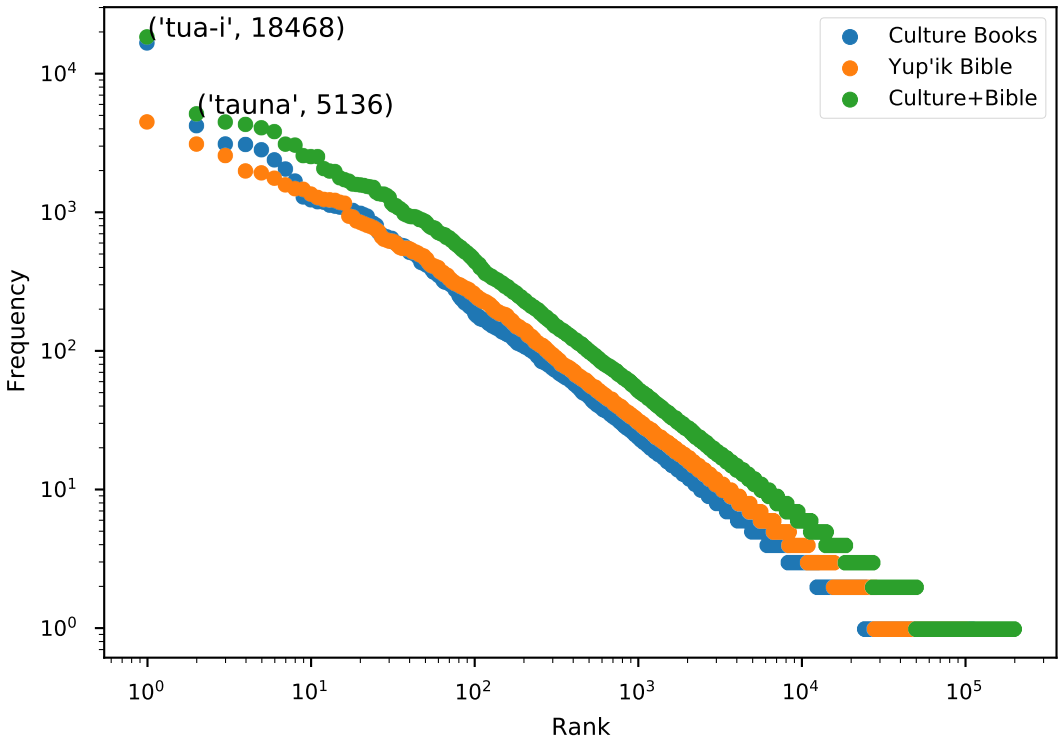

Figure 5.2: Word (Type) Rank vs Frequency (Tokens Per Type)

the Yup'ik Bible, to evaluate the effectiveness of the morphological analyzer. In the next chapter, I will describe the results of these two types of evaluation.

Chapter 6

RESULTS AND ERROR ANALYSIS

In this chapter I provide the results of both of the tests described in Chapter 5, including overall numerical results and error analysis. The numerical results for the test suite described in Section 5.1 are given in Section 6.1. The numerical results for the corpus test described in Section 5.2 are given in Section 6.2 and an error analysis given in Section 6.3.

6.1 Test Suite Results

Section 5.1.1 described the test suite creation of 2,792 test cases. These are the results when run through the generator FST (Table 6.1) and the analyzer FST (Table 6.2).

Key		**Count**	**Percentage**
NO	No generator output	9	0.32%
OI	Only incorrect generator output	99	3.55%
	Total items incorrect generator output	**108**	**3.87%**
UC	Unambiguous, correct generator output	1,530	54.80%
AC	Ambiguous, correct generator output	1,155	41.37%
	Total items correct generator output	**2,685**	**96.17%**

Table 6.1: Generator Test Suite Results

In summary, Tables 6.1 and 6.2 provide the same number of passing tests (96.17%) and failing tests (3.87%). The two tables differ in the percentage of tests that were classified as having either unambiguous output or ambiguous output. The generator FST provided less ambiguous results compared to the analyzer FST with more than half having only the gold answer as a possible output compared to only 18% for the analyzer FST. This may be the result of developing the FST mostly in the generation direction and not on removing ambiguous parses.

Key		Count	Percentage
NO	No analysis	49	1.76%
OI	Only incorrect analysis	59	2.11%
	Total items incorrect analysis	**108**	**3.87%**
UC	Unambiguous, correct analysis	504	18.05%
AC	Ambiguous, correct analysis	2,181	78.12%
	Total items correct analysis	**2,685**	**96.17%**

Table 6.2: Analyzer Test Suite Results

The number of failing tests stayed the same between the generator and analyzer FSTs at 108 tests. The difference being the number of tests that did not parse were higher for the analysis FST. This is reasonable because the generator is receiving valid underlying forms and can provide a guess for a possible surface form. The generator is given valid lexical entries and morphotactics, and can pass through the phonological rules to output a possible answer. The analyzer on the other hand is receiving valid surface forms and because the FST cannot find any valid underlying forms, it does not output anything.

The possible reasons for no analyses from the analyzer could be missing lexical entries, incorrect morphotactics, or missing or incorrectly implemented phonological processes. Future work can be done to extend the coverage and correctly handle the last 108 test cases, but I stopped development with this level of coverage because I didn't have enough time and it was going to take a lot of effort to cover the last 4% of test cases.

These results are very encouraging and show that most of the lexical and grammatical phenomena described in Jacobson's dictionary (2012) and grammar book (1995) are being implemented correctly in the analyzer. This final FST may then be used to empirically show the coverage of phenomena described in the lexicon and grammar on real world text corpora.

6.2 Corpus Test Results

Section 5.2 describes the creation of the corpus test used to evaluate the finite-state morphological analyzer with two different data sources, a corpus of Yup'ik culture books and the

Yup'ik Bible. Since the corpora only provide surface forms and do not provide underlying forms, these tests can only be run through the analysis direction. I will begin this section by providing the parsing coverage of the analyzer over the two corpora in Subsection 6.2.1. Then I will provide figures and tables examining the parse count for all the words in the two corpora in 6.2.2. Finally, I will provide results for the hand classification of a sample of words in 6.2.3.

6.2.1 Parsing Coverage

Parsing coverage measures the number of items for which there are outputs. Table 6.3 shows the number of items with zero and non-zero parses, and calculates coverage as non-zero parses/total items, for both datasets and the combined set. These numbers are provided both at the type and token level.

	Types			**Tokens**		
	Non-zero parses	Zero parses	Coverage	Non-zero parses	Zero parses	Coverage
Culture Books	87,016	21,311	0.803	273,972	33,384	0.891
Yup'ik Bible	87,489	11,183	0.887	297,203	27,517	0.915
Culture+Bible	163,722	31,988	0.837	571,175	60,901	0.904

Table 6.3: Morphological Analyzer Parsing Coverage

In summary, the morphological analyzer performs somewhat better in terms of coverage over the Yup'ik Bible corpus than the culture books corpus. The coverage over the Yup'ik Bible corpus for types and tokens is at 88.7% and 91.5% respectively, and overall the finite-state morphological analyzer provides parses for 83.7% of types and 90.4% of tokens from the combined data sources. Note that the higher token than type coverage means the analyzer has good coverage over high frequency types. The high coverage is also a large indicator that the morphological analyzer developed from a dictionary and grammar book has wide coverage over real world text.

6.2.2 Parse Count

This subsection examines the parse count of the two data sources and provides summaries and correlations in the data. The first figure looks at the spread of parse counts between the two sources and the combined data set in Figure 6.1.

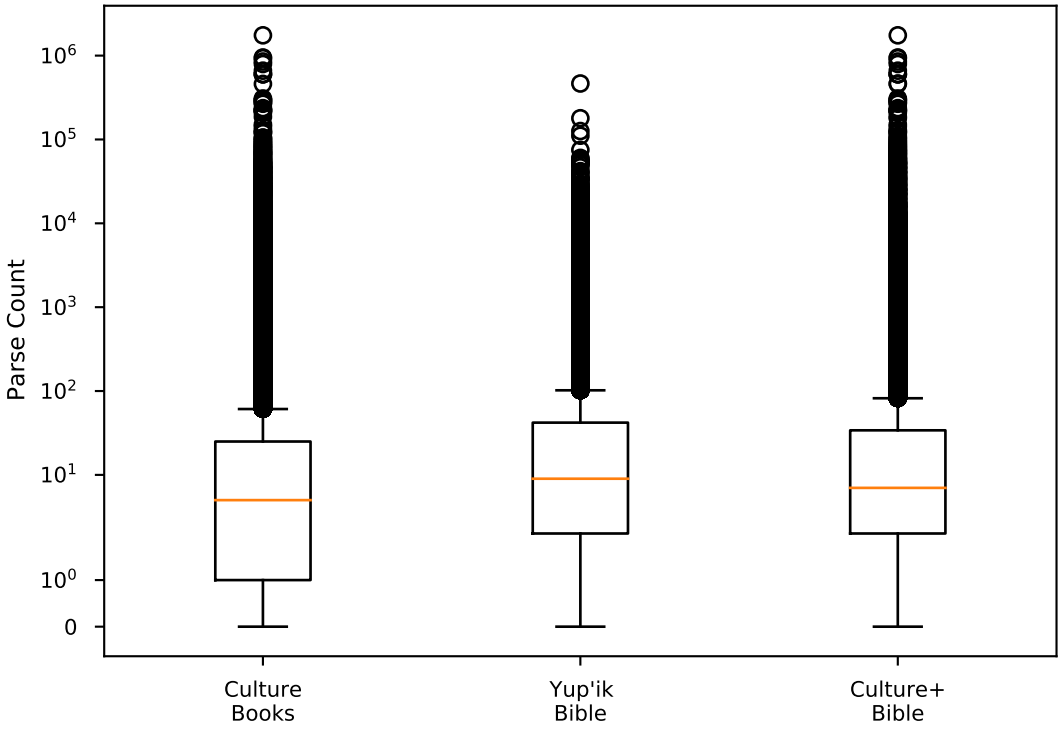

Figure 6.1: Parse Count Spread

The boxplot represents the group of word types and its position gives the number of analyses for that word type, on a log scale. The boxplots in Figure 6.1 show the two corpora and the combined set follow similar patterns. They are similar in size and shape indicating that the parse count distributions and parse count level between the corpora are similar. They show an orange line indicating a median parse count of 5, 9, and 7 for the Culture Books, Yup'ik Bible, and the combined Culture+Bible corpora respectively. The box ranges

Culture Books	Parse Count	Yup'ik Bible	Parse Count
ciunillrullinilriaten	1,746,154	ciuniurilriartangqerquni	464,920
pillrulliniungaqaariitaami	950,040	ciuniurillruciatun	179,124
caqutekassaatullrullinilria	930,960	atawaqercecimalriaruniciqaiceci	126,000
pininrungaitelliniami	850,257	ciuniurilauciicetun	110,920
piniriinanga'artellinilria	794,340	umyuarrlugcarauteksunailucia	75,036
pisciigalillininiartuci	661,520	ciuniuringaitua	60,298
alingitlallrullinilriakut	603,044	calluuciciiqaaci	58,371
qialallrullinilrianga	599,716	yugnikuratullrullinia	56,520
piciqliarutlinikiikuk	597,804	tanqilriakunpiciunrilnguut	54,120
kiarquryaaqaqliniaqelria	459,540	akiilnguiruciciiqniluku	53,786

Table 6.4: Top 10 Most Ambiguous Parses

from a lower quartile of 1 or 2 parses to an upper quartile of around 34 parses. This means that 25% of parse counts range from 1 or 2 to around 7 parses and another 25% range from 7 to 34 parses. The lower whisker indicates that around 25% of words have 0 to 1 or 2 parses. The upper whisker is at around 82 parses to again indicate that around 25% of words fall between 34 and 82 parses. The dots at the top end of the box and whisker plot indicate individual words are outliers. This indicates a few number of words that have an unusually high number of possible analyses.

Table 6.4 shows the top 10 words with the most ambiguous parses in the two different corpora. The Culture Book corpus has by far the word with the most ambiguous parses at 1.7 million possible parses. The Culture Book corpus in general has many words with ambiguous parses but both corpora drop off fairly quickly when looking at words in order of ambiguity. I leave to future work the questions of why these words are ambiguous or what phenomena is causing the large spike in ambiguity.

The next figure, Figure 6.2, examines the correlation between word length and parse count. It has both the Culture Book and Yup'ik Bible corpora as well as the intersection of common words in both corpora plotted in the figure. Figure 6.2 shows that, across both corpora, words that fail to parse can be of any length. It also shows that the common words

between the corpora fall between 3-20 characters in length with the average word length for parsable words is 13.43 and 14.09 characters for the Culture Book and Yup'ik Bible corpora respectively. The longest parsable words in the corpora do not exceed 40 characters in length. While there is no syntactic limit on the length of Yup'ik words, most real world text does not exceed 40 characters.

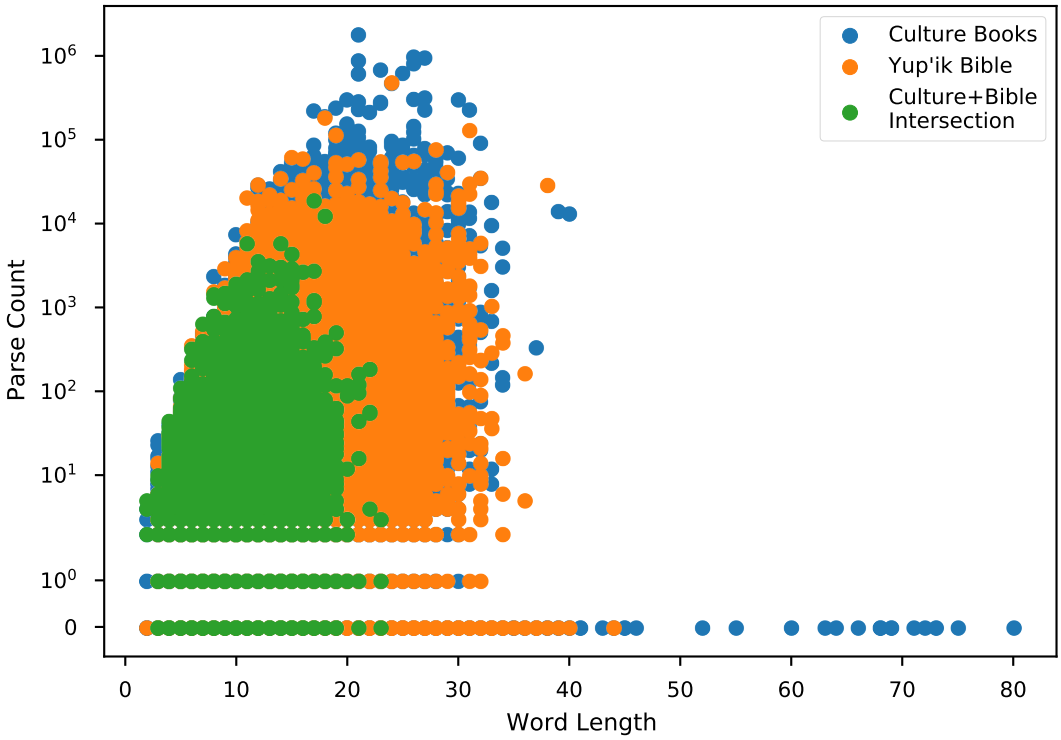

Figure 6.2: Word Length vs Parse Count

6.2.3 Hand Classification Results

As explained in Section 5.2.3, I split the corpus data into 5 "buckets" of around 20k word types each and took a sample of 30 words from each bucket. Then I hand classified each word as defined by Table 5.3, and the results are displayed in Tables 6.5 and 6.6.

		Classification						
		Unknown	Incorrect			Correct		
		UN	NO	OI	OG	UC	AC	**Correct/Total**
Non-zero Parses (Most Ambiguous - Least Ambiguous)	First Quarter	2		3	1		24	**0.80**
	Second Quarter			1	2		27	**0.90**
	Third Quarter			2	2		28	**0.87**
	Fourth Quarter			2		14	14	**0.93**
Zero Parses		6	24					**0.00**
Total		**8**	**24**	**8**	**5**	**14**	**91**	**0.70**

Table 6.5: Classification Results - Culture Books

		Classification						
		Unknown	Incorrect			Correct		
		UN	NO	OI	OG	UC	AC	**Correct/Total**
Non-zero Parses (Most Ambiguous - Least Ambiguous)	First Quarter			5			25	**0.83**
	Second Quarter			5			25	**0.83**
	Third Quarter			3			27	**0.90**
	Fourth Quarter			3		10	17	**0.90**
Zero Parses			30					**0.00**
Total		**0**	**30**	**16**		**10**	**94**	**0.69**

Table 6.6: Hand Classification Results - Yup'ik Bible

In the last columns titled "Correct/Total" of Tables 6.5 and 6.6, I list the percentage of the words that had the correct parse in the output. The words classified as unknown are included in the total for a more conservative result. For the non-zero parses rows, the analyzer has the correct parse for 80-90% of the words. The zero parses row, of course, has no correct parses.

The reason for splitting the data into "buckets" was to determine if there is any correlation between words with more ambiguity vs less ambiguity having any differences in classification types, but since the non-zero parses rows are similar in classification counts and correct percentage between the two corpora, the outcome is a clear no.

In summary, since the "buckets" were of semi-equal size and an equal random sample between the buckets and corpora were selected, the final total of 70% correct can be used to

estimate the accuracy of the CAY finite-state morphological analyzer.

6.3 Corpus Test Error Analysis

In this section I further hand classified the incorrect words in the corpus test as defined by the error classification in Table 5.4. The results in Table 6.7 show the error count for each of the classification types between the two corpora.

Error Key	Culture	Bible
X.PRG.LEX	9	10
X.PRG.TAC	1	2
X.PRG.PHO	7	28
X.MIS.ORT	5	1
X.MIS.OCR	12	0
X.MIS.PRE	0	1
X.CMX.ENG	3	4
Total	**37**	**46**

Table 6.7: Error Analysis Results

Table 6.7 shows that there were a near equal amount of errors due to missing lexical entries and morphotactics, but there were way more errors due to morphophonology in the Yup'ik Bible corpus. Since the Culture Book corpus was scanned through OCR, there are many more of that word error type. There also were nearly an equal amount of English code-mixing errors in the two corpora. In total, there were many errors that could have been averted with better OCR and removal of English words, but there were still many program errors due to missing lexical entries and incorrect implementations of morphophonological processes. These can be addressed by incrementally improving the morphological analyzer in future work.

6.4 Summary

This chapter has presented the results and error analysis for both the test suite and the corpus test. The test suite created for my development set has shown to correctly generate

and analyze 96% of the words described in Jacobson's (1995) grammar book. The corpus test has shown a parsing coverage of between 83.7-90.4% for both the Culture Book and Yup'ik Bible corpora. And finally, the hand classification results estimate a 70% accuracy of the CAY finite-state morphological analyzer. In the next chapter, I will discuss what I learned and future work for the CAY finite-state morphological analyzer.

Chapter 7

CONCLUSION AND FUTURE WORK

This thesis describes the creation and evaluation of a finite-state morphological analyzer for Central Alaskan Yup'ik. A functional practical language technology can be made by using just a few resources: a dictionary, a descriptive grammar, and a small corpus of text to evaluate on.

In this thesis, I began with an overview of CAY in Chapter 2 and finite-state morphology in Chapter 3, the two topics necessary to create a finite-state morphological analyzer. In Chapter 4, I described the design of the analyzer which included the lexicon, underlying form representation, and the components of the FST stack. In Chapter 5, I described the test suite and the corpus test and finally, in Chapter 6, I gave the results of the two tests and an error analysis of the corpus test results.

In the process of this project, I have gained a much deeper understanding of the morphology of CAY and the finite-state tools. I learned a lot from having to understand the intricacies of each of the morphophonological processes and decide which formalism was the best choice for implementing the process. In addition to the morphophonology, I gained insight into the complexity of the morphosyntax and morphosemantics of the language. Furthermore, many of the errors classified as unknown in the error analysis may stem from morphological phenomena not described in the descriptive grammar which provide fertile ground for further research.

In addition to cutting down analysis ambiguity and program errors, there is still a lot of functionality that can be implemented into the program. These include adding dialectal variants to the lexicon and the ability to specify which dialect the analyzer should reflect. There is also the need to add the ability for code-mixing of English bases with CAY inflection

since they occur quite frequently in corpus text. It may also be beneficial to implement some sort of verb valency and valency-changing functionality, possibly by adding weights into the FST.

I am excited for the future work of integrating this morphological analyzer into other language technologies such as an intelligent dictionary, spell-checker, intelligent CALL software, and syntactic/semantic parsers. Additionally, the morphological analyzer may be used in creating a morphologically tagged corpus which may be used to make a language model by adding weights to the analyzer. I hope that this finite-state morphological analyzer and future tools may be beneficial to the Central Alaskan Yup'ik community and future language learners.

BIBLIOGRAPHY

Allauzen, Cyril, Riley, Michael, Schalkwyk, Johan, Skut, Wojciech, and Mohri, Mehryar. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata*, pages 11–23. Springer.

American Bible Society. 2014. *Tanqilriit Igat*. American Bible Society, New York, NY.

Andriyanets, Vasilisa and Tyers, Francis. 2018. A prototype finite-state morphological analyser for Chukchi. In *Proceedings of the Workshop on Computational Modeling of Polysynthetic Languages*, pages 31–40, Santa Fe, New Mexico, USA, 2018. Association for Computational Linguistics.

Arppe, Antti, Lachler, Jordan, Trosterud, Trond, Antonsen, Lene, and Moshagen, Sjur N. 2016. Basic language resource kits for endangered languages: A case study of Plains Cree. In *Proceedings of the 2nd Workshop on Collaboration and Computing for Under-Resourced Languages Workshop (CCURL 2016), Portorož, Slovenia*, pages 1–8.

Arppe, Antti, Cox, Christopher, Hulden, Mans, Lachler, Jordan, Moshagen, Sjur N, Silfverberg, Miikka, and Trosterud, Trond. 2017a. Computational modeling of verbs in Dene languages: The case of Tsuut'ina. *Working Papers in Athabaskan (Dene) Languages*, pages 51–69.

Arppe, Antti, Junker, Marie-Odile, and Torkornoo, Delasie. 2017b. Converting a comprehensive lexical database into a computational model: The case of East Cree verb inflection. In *Proceedings of the 2nd Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 52–56, Honolulu, 2017b. Association for Computational Linguistics.

Assini, Alicia Alexandra. 2013. *Natural Language Processing and the Mohawk Language: Creating a Finite State Morphological Parser of Mohawk formal nouns*. PhD thesis, University of Limerick.

Beesley, Kenneth R. and Karttunen, Lauri. 2003a. *Finite State Morphology*. CSLI studies in computational linguistics: Center for the Study of Language and Information. CSLI Publications.

Beesley, Kenneth R and Karttunen, Lauri. 2003b. *Two-level Rule Compiler*. Xerox Corporation, Palo Alto Research Center.

Bills, Aric, Levin, Lori S, Kaplan, Lawrence D, and MacLean, Edna Ahgeak. 2010. Finite-state morphology for Iñupiaq. In *7th SaLTMiL Workshop on Creation and use of basic lexical resources for less-resourced languages LREC 2010, Valetta, Malta, 23 May 2010 Workshop programme*, page 19.

Bowers, Dustin, Arppe, Antti, Lachler, Jordan, Moshagen, Sjur, and Trosterud, Trond. 2017. A morphological parser for Odawa. In *Proceedings of the 2nd Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 1–9, Honolulu, 2017. Association for Computational Linguistics.

Chen, Emily and Schwartz, Lane. 2018. A morphological analyzer for St. Lawrence Island / Central Siberian Yupik. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, 2018. European Language Resources Association (ELRA).

Chen, Emily, Park, Hyunji Hayley, and Schwartz, Lane. 2020. Improved finite-state morphological analysis for St. Lawrence Island Yupik using paradigm function morphology. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 2676–2684, Marseille, France, 2020. European Language Resources Association.

Chomsky, Noam and Halle, Morris. 1968. *The Sound Pattern of English*. Harper and Row, New York.

Curtis, Christian. 2014. A finite-state morphological analyzer for a Lakota HPSG grammar. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 541–544, Reykjavik, Iceland, 2014. European Language Resources Association (ELRA).

Enoch, Charles. December 2015. Holy Bible translated into modern Yup'ik. *Alaska Public Media*. URL https://www.alaskapublic.org/2015/12/30/holy-bible-translated-into-modern-yupik/.

Fienup-Riordan, Ann and Rearden, Alice. 2011. *Qaluyaarmiuni Nunamtenek Qanemciput = Our Nelson Island Stories: Meanings of Place on the Bering Sea Coast*. Calista Elders Council in association with University of Washington Press, Seattle, WA.

Fienup-Riordan, Ann and Rearden, Alice. 2013. *Erinaput Unguvaniartut = So Our Voices Will Live: Quinhagak History and Oral Traditions*. Calista Elders Council and the Alaska Native Language Center, University of Alaska Fairbanks, Fairbanks, AK.

Fienup-Riordan, Ann and Rearden, Alice. 2014. *Nunamta Ellamta-llu Ayuqucia = What Our Land and World Are Like: Lower Yukon History and Oral Traditions*. Calista Elders Council and Alaska Native Language Center, Fairbanks, AK.

Fienup-Riordan, Ann and Rearden, Alice. 2016a. *Anguyiim Nalliini = Time of Warring: The History of Bow-and-Arrow Warfare in Southwest Alaska*. University of Alaska Press, Fairbanks, AK.

Fienup-Riordan, Ann and Rearden, Alice. 2016b. *Ciulirnerunak Yuuyaqunak = Do Not Live Without an Elder: The Subsistence Way of Life in Southwest Alaska*. University of Alaska Press, Fairbanks, AK.

Fienup-Riordan, Ann, Meade, Marie, and Rearden, Alice. 2017. *Qanemcit Amllertut = Many Stories to Tell: Tales of Humans and Animals in Southwest Alaska*. University of Alaska Press, Fairbanks, AK.

Fienup-Riordan, Ann, Meade, Marie, and Rearden, Alice. 2018. *Yupiit Qanruyutait = Yup'ik Words of Wisdom*. University of Nebraska Press, Lincoln, NE.

Graham, Dougal. 2007. Finite-state parsing of Cayuga morphology. Master's thesis, Memorial University of Newfoundland (Canada), Canada.

Harrigan, Atticus G, Schmirler, Katherine, Arppe, Antti, Antonsen, Lene, Trosterud, Trond, and Wolvengrey, Arok. 2017. Learning from the computational modelling of Plains Cree verbs. *Morphology*, 27(4):565–598.

Hensel, Chase, Blanchett, Marie, Alexie, Ida, and Morrow, Phyllis. 1985. *Qaneryaurci Yup'igtun = Learn to speak Yup'ik*. Yup'ik Language Center; Kuskokwim Community College, Bethel, AK.

Hulden, Mans. 2009a. *Finite-state machine construction methods and algorithms for phonology and morphology*. Ph.D., The University of Arizona, Tucson, AZ.

Hulden, Mans. 2009b. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 29–32, Athens, Greece, 2009b. Association for Computational Linguistics.

Institute for Information Technology. 2012. The UQAILAUT Project: Inuktitut Morphological Analyzer. URL http://inuktitutcomputing.ca/Uqailaut/info.php.

Jacobson, Steven A. 1984. *Yup'ik Eskimo Dictionary*. Alaska Native Language Center, University of Alaska, Fairbanks, AK, 1st edition.

Jacobson, Steven A. 1995. *A Practical Grammar of the Central Alaskan Yup'ik Eskimo*

Language. Alaska Native Language Center and Program, University of Alaska, Fairbanks, AK.

Jacobson, Steven A. 2012. *Yup'ik Eskimo Dictionary*. Alaska Native Language Center, University of Alaska Fairbanks, Fairbanks, AK, 2nd edition.

John, Paul, Fienup-Riordan, Ann, and Sheild, Sophie. 2003. *Qulirat Qanemcit-llu Kinguvarcimalriit = Stories for Future Generations: The Oratory of Yup'ik Paul John*. Calista Elders Council in association with University of Washington Press, Seattle, WA.

Johnson, Ryan, Antonsen, Lene, and Trosterud, Trond. 2013. Using finite state transducers for making efficient reading comprehension dictionaries. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013)*, pages 59–71.

Kaplan, Ronald M and Kay, Martin. 1981. Phonological rules and finite-state transducers. In *Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting*, pages 27–30.

Kaplan, Ronald M. and Kay, Martin. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karttunen, Lauri. Finite-state lexicon compiler. Technical Report ISTL-NLTT- 1993-04-02, Xerox Corporation, Palo Alto Research Center, Palo Alto, CA, 1993.

Karttunen, Lauri. 1994. Constructing lexical transducers. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*.

Karttunen, Lauri. June 1995. The replace operator. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics.

Karttunen, Lauri and Beesley, Kenneth R. Two-level rule compiler. Technical Report ISTL-92-2, Xerox Corporation, Palo Alto Research Center, Palo Alto, CA, 1992.

Karttunen, Lauri, Koskenniemi, Kimmo, Kaplan, Ronald, and others. 1987. A compiler for two-level phonological rules. *Tools for morphological analysis*.

Karttunen, Lauri, Kaplan, Ronald M., and Zaenen, Annie. 1992. Two-level morphology with composition. In *COLING 1992 Volume 1: The 15th International Conference on Computational Linguistics*.

Kazeminejad, Ghazaleh, Cowell, Andrew, and Hulden, Mans. 2017. Creating lexical resources for polysynthetic languages—the case of Arapaho. In *Proceedings of the 2nd Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 10–18, Honolulu, 2017. Association for Computational Linguistics.

Koskenniemi, Kimmo. 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*, volume no. 11 of *Publications / Department of General Linguistics. University of Helsinki*. University of Helsinki.

Lachler, Jordan, Antonsen, Lene, Trosterud, Trond, Moshagen, Sjur, and Arppe, Antti. 2018. Modeling Northern Haida verb morphology. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, 2018. European Language Resources Association (ELRA).

Lindén, Krister and Pirinen, Tommi A. 2009. Weighted finite-state morphological analysis of Finnish compounding with HFST-LEXC. In *Proceedings of the 17th Nordic Conference of Computational Linguistics (NODALIDA 2009)*, pages 89–95.

Lindén, Krister, Axelson, Erik, Hardwick, Sam, Pirinen, Tommi A., and Silfverberg, Miikka. 2011. HFST—framework for compiling and applying morphologies. In Mahlow, Cerstin and Piotrowski, Michael, editors, *Systems and Frameworks for Computational Morphology*, pages 67–85, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

Lovick, Olga, Cox, Christopher, Silfverberg, Miikka, Arppe, Antti, and Hulden, Mans. May 2018. A computational architecture for the morphology of Upper Tanana. In *Proceedings*

of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

Mithun, Marianne. 2000. Valency-changing derivation in Central Alaskan Yup'ik. *Changing valency: case studies in transitivity*, pages 84–114. Publisher: Cambridge Univ Pr.

Mithun, Marianne. 2012. Exuberant complexity: The interplay of morphology, syntax, and prosody in Central Alaskan Yup'ik. *Linguistic Discovery*, 10.

Miyaoka, Osahito. 2012. *A Grammar of Central Alaskan Yup'ik (CAY)*. Mouton grammar library ; 58. De Gruyter Mouton, Berlin.

Miyaoka, Osahito and Mather, Elsie. 1979. *Yup'ik Eskimo Orthography*. Yup'ik Language Center; Kuskokwim Community College, Bethel, AK, 2nd edition.

Moshagen, Sjur N., Pirinen, Tommi, and Trosterud, Trond. May 2013. Building an open-source development infrastructure for language technology projects. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013)*, pages 343–352, Oslo, Norway, May 2013. Linköping University Electronic Press, Sweden.

Oqaasileriffik. 2010. A Bit of History. URL https://oqaasileriffik.gl/langtech/a-bit-of-history/.

Reed, Irene, Miyaoka, Osahito, Jacobson, Steven A, Afcan, Paschal, and Krauss, Michael. 1977. *Yup'ik Eskimo Grammar*. Alaska Native Language Center and Yup'ik Language Workshop, Fairbanks, AK.

Rios, Annette. 2015. *A basic language technology toolkit for Quechua*. PhD Thesis, University of Zurich.

Schmid, Helmut. 2005. A programming language for finite state transducers. In *Finite-State Methods and Natural Language Processing FSMNLP 2005*, page 50.

Schwartz, Lane, Chen, Emily, Hunt, Benjamin, and Schreiner, Sylvia L.R. 2019. Bootstrapping a neural morphological analyzer for St. Lawrence Island Yupik from a finite-state transducer. In *Proceedings of the 3rd Workshop on the Use of Computational Methods in the Study of Endangered Languages Volume 1 (Papers)*, pages 87–96, Honolulu, 2019. Association for Computational Linguistics.

Snoek, Conor, Thunder, Dorothy, Lõo, Kaidi, Arppe, Antti, Lachler, Jordan, Moshagen, Sjur, and Trosterud, Trond. 2014. Modeling the noun morphology of Plains Cree. In *Proceedings of the 2014 Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 34–42, Baltimore, Maryland, USA, 2014. Association for Computational Linguistics.

Solórzano, Sofía Flores. 2017. Desarrollo de un analizador automático de estados finitos para la lengua bribri. URL http://morphology.bribri.net.

Tennant, Edward A. and Bitar, Joseph N. 2000. *Yuut Qanemciit: Yupiit Cayaraita Qanrutkumallrit = Yup'ik Lore: Oral Traditions of an Eskimo People.* Lower Kuskokwim School District, Bethel, AK.

Trosterud, Trond. 2006. Grammatically based language technology for minority languages. *Lesser-Known languages of South Asia: Status and policies, case studies and applications of information technology*, pages 293–316.

Woodbury, Anthony. 1981. *Study of the Chevak Dialect of Central Yup'ik Eskimo.* ProQuest Dissertations Publishing.

Woodbury, Anthony C. June 2014. Morphological orthodoxy in Yupik-Inuit. *Annual Meeting of the Berkeley Linguistics Society*, 30(2).

Çöltekin, Çağrı. 2010. A freely available morphological analyzer for Turkish. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, pages 820–827.

Appendix A

MORPHOPHONOLOGICAL SYMBOLS AND SPECIAL CHARACTERS

+	keep final consonants of bases
– (minus)	drop final consonants of bases
~	drop final *e* from bases
%	keep "strong" final consonants, drop "weak" final consonants
:	drop voiced velar continuant if they occur between single vowels
' (apostrophe)	suffix causes gemination for bases of the form (C)VC*e*-
.	no change to either base or suffix at the juncture
@	*te* ending bases dependent suffix
@[1]	(*@n* postbase) drops *t* only when the *t* is preceded by a fricative; the fricative is then written devoiced
@[2]	(*@i/m/ng/t/v*) drops *t* when the *t* is preceded by a fricative which is then written devoiced, and changes *t* to *s* (*y* in HBC) when the *t* is preceded by a vowel, unless the base is marked with °, in which case *t* is changed to *l*
@[3]	(*@(u)c/t*) drops *t* when the *t* is preceded by a fricative which then becomes voiced, and changes *t* to *y* if preceded by a vowel, but to *s* (except in HBC) if the *t* is geminated on the base, unless the base is marked with °, in which case *t* is changed to *l*
@[4]	(*@ng/k/t*) changes *t* to *s* (*y* in HBC) whether preceded by a vowel or fricative, unless the base is marked with °, in whcih case t is changed to *l*
@[5]	(*@c/l/ll/k/ng/p/t*) drops any *t*

@^{l}	(*@l* postbases) *tl* → *ll*
@^{y}	(*@y* postbases) *ty* → *c*
@^{ʕ}	(*@ur* postbases) *te* → *q*
∗	base final *r* marked with asterisk to indicate "strong" base
°	special *te*
(ar) (ar*)	*ar*-bases indicates that this segment of the word or base is deleted if it occurs at the end of the word, or before a consonant-retaining suffix that starts with a consonant
(ur)	*ur*-bases (only in *–r(ur)lur*) – same as *(ar)/(ar*)*
(e)	is used only when a strong base-final consonant is kept
(g)	is used with bases ending in two vowels
(ng)	is used with bases ending in a vowel
(r)	is used with bases ending in *te*, and by some speakers in *e* not preceded by *t*
(s)	is used with bases ending in a vowel
(t)	is used with bases ending in a consonant
(u)	is used with bases ending in a consonant or *e* (includes *te* bases)
(g/t)	*(g)(t)*
(r/l)	*@+(r/l)i-*[1] postbase - 'to become or cause to become V' # used with adjectival verbs; the *r* is used with bases ending in a prime vowel, and the *l* with bases ending in *e* (but (special) *te* changes to *l*)
(u/i)	consonant and *e*-ending bases (not *te* bases) use *u* ; *te* bases use *i*
k̲ q̲ g̲ r̲ gg̲ rr̲	velar/uvular (front/back velar) assimilation
[e]	weak *e* initial bases - optional *e* used for prosody
a̤ ṳ	vowels from *(ar)* or *(ar*)*, or *(ur)* respectively (FST internal representation)

â î û ê ậ ụ̂	indicates rhythmic lengthening of the vowel (and correspondingly stress on the syllable) (FST internal representation)
á í ú é ậ ụ́	indicates stress of the syllable (FST internal representation)

Appendix B

ESU.LEXC FILE (EXCERPT)

```
Multichar_Symbols
vv ll ss gg rr ng ng ug ugg ur urr m̄ n̄
[Particle]
[N→N]  [N→V]  [V→N]  [V→V]
[N] [Abs] [Unpd] [Sg] [3SgPoss] [SgPosd]
[V] [Intr] [Ind] [S_3Sg] [A_3Sg] [P_3Sg]
[Encl]

LEXICON Root
Particle;
NounBase;
VerbBase;

LEXICON Particle
aa=ang                  ParticleEnd; ! yes; you're welcome

LEXICON ParticleEnd
[Particle]:             EncliticOrEnd;

LEXICON NounBase
angyar                  NounPostbase; ! boat

LEXICON VerbBase
aqvate                  VerbPostbase; ! to go get something

LEXICON NounPostbase
NounInflection;
--cuar(ar*)[N→N]:-cuar(ar*)  NounPostbase; ! small N; tiny N; little N
-%:~(ng)u[N→V]:%:~(ng)u      VerbPostbase; ! to be N

LEXICON VerbPostbase
VerbInflection;
--qatar[V→V]:-qatar          VerbPostbase; ! to be about to V
-@~+vig[V→N]:@²~+vig         NounPostbase; ! place to V
```

```
LEXICON NounInflection
[N][Abs][Unpd][Sg]:0                EncliticOrEnd;
[N][Abs][3SgPoss][SgPosd]:%:(ng)a   EncliticOrEnd;

LEXICON VerbInflection
[V][Intr][Ind]:+'(g/t)u IntransitiveIndicative;
[V][Trns][Ind]:+'(g)a TransitiveIndicative;

LEXICON IntransitiveIndicative
[S_3Sg]:q              EncliticOrEnd;

LEXICON TransitiveIndicative
[A_3Sg][P_3Sg]:a       EncliticOrEnd;

LEXICON EncliticOrEnd
#;
=llu[Encl]:=llu        EncliticOrEnd; ! and; also
```

Appendix C

ESU.LEXC.XFST FILE (EXCERPT)

```
define Stop       [ p | t | c | k | q | k̲ | q̲ ];
define VoiceFric  [ v | l | s | g | r | ug | ur | g̲ | r̲ | ġ ];
define VlessFric  [ vv | ll | ss | gg | rr | ugg | urr | g̲g̲ | r̲r̲ ];
define VoiceNasal [ m | n | ng ];
define VlessNasal [ m̄ | n̄ | ng ];

define VoiceCon   [ VoiceFric | VoiceNasal ];
define VlessCon   [ VlessFric | VlessNasal | Stop ];

define Fric       [ VlessFric  | VoiceFric  ];
define Nasal      [ VlessNasal | VoiceNasal ];

define Con        [ Stop | Fric | Nasal | w | y ];

define FVow       [ a | i | u | ạ | ụ ];
define Vow        [ e | FVow ];

define Lowercase "A" -> a, "C" -> c, "E" -> e, "G" -> g, "I" -> i, "K" -> k,
"L" -> l, "M" -> m, "N" -> n, "P" -> p, "Q" -> q, "R" -> r, "S" -> s, "T" -> t,
"U" -> u, "V" -> v, "W" -> w, "Y" -> y, "M̄" -> m̄, "N̄" -> n̄, "Vv" -> vv, "Ll" -> ll,
"Ss" -> ss, "Gg" -> gg, "Rr" -> rr, "Ng" -> ng, "Ng" -> ng, "Ug" -> ug, "Ur" -> ur,
"Ugg" -> ugg, "Urr" -> urr;


define arExpansion "(ar)"  -> "ạ" r       .o.
   "(ar*)" -> "ạ" r "*"    ;

define MultipleForms t e "-r̲pag/@²vag" (->) l "." v a g || Vow _     .o.
"-r̲pag/@²vag"  ->  "-" r̲ p a g                                        .o.

"-r̲(ur)lur" -> "+" ụ r l u r      || g _                              .o.
"-r̲(ur)lur" -> "-" "r̲" ụ r l u r                                      .o.

"@+cete/.vkar" -> "@" "+" c e t e      || Con _                       .o.
```

```
"@+cete/.vkar" (->) "@" "+" c e t e    || t e ("°") _                  .o.
"@+cete/.vkar" -> "." v k a r                                          .o.

"+ciqe/@ciiqe" -> "@" c i i q e        || t e ("°") _                  .o.
"+ciqe/@ciiqe" -> "+" c i q e                                          .o.

t e ("°") "@+-'(g)ar(ar)te" -> g "+" a r "ạ" r t e    || Vow _          .o.
t e ("°") "@+-'(g)ar(ar)te" -> "+" a r "ạ" r t e      || Con _          .o.
t "'" e "@+-'(g)ar(ar)te" -> s "'" "+" a r "ạ" r t e                   .o.
"@+-'(g)ar(ar)te" -> "-" a r "ạ" r t e                || r _           .o.
"@+-'(g)ar(ar)te" -> "+" a r "ạ" r t e                || g _           .o.
"@+-'(g)ar(ar)te" -> "+" "'" "(g)" a r "ạ" r t e                       .o.

"@~+lu/@na" -> "@" n a                || "°" _                        .o.
"@~+lu/@na" -> "@" "~" "+" l u                                         .o.
"[STE_2Sg]" -> k                       || n a _                        .o.
"[STE_2Sg]" -> t e n                                                   .o.
i (->) u                               || \[i] _ t e "°" "@" n a       .o.

"-lria/@+ngur*" (->) "@" "+" ng u r "*"  || t e "°" _
"-lria/@+ngur*" -> "-" [l|ll] r i i ||
[Stop|VlessCon] ("'") e ("°") ([r|rr|g]) _ "%" ":" "(e)" .o.
"-lria/@+ngur*" -> "-" l r i i || _ "%" ":" "(e)"
"-lria/@+ngur*" -> "-" [l|ll] r i a || [Stop|VlessCon] ("'") e ("°") ([r|rr|g]) _
"-lria/@+ngur*" -> "-" l r i a

"[Opt_PRS_S_2Sg]" -> "~" "(g)" i       || [ Vow Vow | [ Con - t ] e ] _  .o.
"[Opt_PRS_S_2Sg]" -> "+" n             || t e _                          .o.
t e "°" "[Opt_PRS_S_2Sg]" -> l u                                         .o.
"[Opt_PRS_S_2Sg]" -> ":" a             || Con _                          .o.
"[Opt_PRS_S_2Sg]" -> 0                                                   .o.
"[Opt_PRS_S_2PlDu]" (->) "~" "(g)" i   || [ Vow Vow | [ Con - t ] e ] _  .o.
"[Opt_PRS_S_2PlDu]" -> "@" "+"                                           .o.
"[Opt_PRS_A_2Sg]" (->) "~" "(g)" i     || [ Vow Vow | [ Con - t ] e ] _  .o.
"[Opt_PRS_A_2Sg]" -> "@" "+"                                             .o.
"[Opt_PRS_A_2Sg_P_3Sg]" -> "~" "(g)" i u || [ Vow Vow | [ Con - t ] e ] _ .o.
t e "[Opt_PRS_A_2Sg_P_3Sg]" -> s g u                                     .o.
t e "°" "[Opt_PRS_A_2Sg_P_3Sg]" -> l g u                                 .o.
"[Opt_PRS_A_2Sg_P_3Sg]" -> "-" gg u    || g _                            .o.
"[Opt_PRS_A_2Sg_P_3Sg]" -> "-" rr u    || r _                            .o.
"[Opt_PRS_A_2Sg_P_3Sg]" -> u                                             .o.
"[Opt_PRS_A_2Sg_P_1Sg]" (->) "~" "(g)" i a || [ Vow Vow | [ Con - t ] e ] _ .o.
"[Opt_PRS_A_2Sg_P_1Sg]" -> "@" "+" ng a                                  .o.
```

```
"@+p/~v" -> "@" "+" p                  || [ t e ("°") | Con ] _        .o.
"@+p/~v" -> "~" v                                                      .o.
"+p/.v" -> "+" p e                      || Con _ Con                   .o.
"+p/.v" -> "+" p                        || Con _ Vow                   .o.
"+p/.v" -> "." v                                                       .o.
"+pegun/.vkun" -> "+" p e g u n         || Con _                       .o.
"+pegun/.vkun" -> "." v k u n                                          .o.
"+t/.s" -> "+" t                        || Con _                       .o.
"+t/.s" -> "." s                                                       .o.
"+c/.s" -> "+" c                        || Con _                       .o.
"+c/.s" -> "." s                                                       .o.
"+c/.ss" -> "." ss                      || Vow _                       .o.
"+c/.ss" -> "+" c                                                      .o.
"('a)" -> ' a                           || \.#. Con a _                .o.
"('a)" -> 0                                                             ;


define AutomaticDevoicing v -> vv, l -> ll, s -> ss, g -> gg, g -> gg,
r -> rr, r̲ -> r̲r̲,  ug -> ugg, ur -> urr ||
[ Stop | VlessFric ] _  , _ [ Stop ] .o.
m -> m̄, n -> n̄, ng -> ng || [ Stop | VlessFric ] _   ;

! Unnecessary devoicing
!                         r -> rr  || _ .#.                            .o.
!                         s -> ss  || .#. _                             ;

regex Lowercase          .o.
      arExpansion        .o.
      MultipleForms      .o.
      AutomaticDevoicing  ;
```

Appendix D

ESU.TWOL FILE (EXCERPT)

```
Alphabet
    a c e g i k l m n p q r s t u v w y m̄ n̄
    vv ll ss gg rr ng ng ug ugg ur urr
    k̲:k q̲:q g:g r̲:r gg:gg r̲r̲:rr
    %[e%] %[e%]:e
    '
    ạ ụ ġ

    ! All operators and allomorph operators map to morpheme segmenter
    %(e%):%> %(g%):%> %(r%):%> %(s%):%> %(t%):%> %(u%):%>
    %(ar%):%> %(ar%*%):%> %(ng%):%>
    %(g%/t%):%> %(r%/l%):%> %(u%/i%):%>

    %+:%> %-:%> %~:%> %%:%> %::%> %':%> %.:%> %*:%> %°:%>
    %@¹:%> %@²:%> %@³:%> %@ :%> %@ :%> %@ :%> %@ :%> %@ :%> %@:%>
    %=;

Sets
    Bndry = %+ %- %~ %% %: %' %. %* %°
        %@¹ %@² %@³ %@ %@ %@ %@ %@ %@
        %(a%) %(e%) %(g%) %(r%) %(s%) %(t%) %(u%)
        %(aa%) %(ar%) %(ar%*%) %(ng%) %(ur%) %(g%/t%) %(r%/l%) %(u%/i%) ;
    AtSign = %@¹ %@² %@³ %@ %@ %@ %@ %@ %@ ;

    Stop = p t c k q k̲ q̲ ;
    VoiceFric = v l s g r ug ur g r̲ ġ ;
    VlessFric = vv ll ss gg rr ugg urr gg r̲r̲ ;
    VoiceNasal = m n ng ;
    VlessNasal = m̄ n̄ ng ;
    VoiceCon = VoiceFric VoiceNasal ;
    VlessCon = VlessFric VlessNasal Stop ;
    Fric = VoiceFric VlessFric ;
    Nasal = VoiceNasal VlessNasal ;
    Con = Stop Fric Nasal w y ;
```

```
    FVow = a i u ạ ụ ;
    Vow = FVow e ;

Rules

"a:0 = --"
a:0 <=> _ Con:0 (%*:) %-: %-: l:0 ;

"a:i = age/ii | aga/ii | anga/ii | aga/ii | enga/ii"
a:i <=> [:Con|.#.] _ g: %%: %:: %(e%):i Con: ;
        [:Con|.#.] _ g: %:: (%~:) (%(ng%):) [a:|ạ:] [Bndry:|Con:|.#.] ;
        [:Con|.#.] _ %:: [ng:|%(ng%):] [a:|ạ:] [Bndry:|Con:|.#.] ;

        [:Con|.#.] [a:|ạ:] g: %:: (%~:) (%(ng%):) _ [Bndry:|Con:|.#.] ;
        [:Con|.#.] [a:|ạ:|e:] %:: [ng:|%(ng%):] _ [Bndry:|Con:|.#.] ;

    except
        _ Con:0 (%*:) %-: %-: l:0 ;

"ạ:0 = --"
ạ:0 <=> _ Con:0 (%*:) %-: %-: l:0 ;

"ạ:i = age/ii | aga/ii | anga/ii | aga/ii | enga/ii"
ạ:i <=> [:Con|.#.] _ g: %%: %:: %(e%):i Con: ;
        [:Con|.#.] _ g: %:: (%~:) (%(ng%):) [a:|ạ:] [Bndry:|Con:|.#.] ;
        [:Con|.#.] _ %:: [ng:|%(ng%):] [a:|ạ:] [Bndry:|Con:|.#.] ;

        [:Con|.#.] [a:|ạ:] g: %:: (%~:) (%(ng%):) _ [Bndry:|Con:|.#.] ;
        [:Con|.#.] [a:|ạ:|e:] %:: [ng:|%(ng%):] _ [Bndry:|Con:|.#.] ;
    except
        _ Con:0 (%*:) %-: %-: l:0 ;

"e:0 = ~ | word# | eVowPostbase | semifinaleVowPostbase | (e):e | -- | te@ "
e:0 <=> _ [ Bndry: - %': - %(r%): ]* %~: ;
        :Vow [Bndry:%>|:0]* t:n _ [ .#. | %= ] ;
        _ Bndry:+ :Vow ;
        .#. ~[ :* :Con :VlessCon] _ [ r | g ] [ Bndry: - %(t%): - %(g%/t%): ]+ :Vow ;
        [[Vow Con]|[Con VoiceCon]] _ [ r | g ] Bndry:* %%:%> Bndry:* %(e%):e ;
        _ (Con:0) Bndry:* %-: %-: l:0 ;
        t: _ [Bndry:-AtSign:]* AtSign: ;
    except
        [:Con|.#.] _ %:: [ng:|%(ng%):] [i:|u:] [Bndry:|Con:|.#.] ;
        [:Con|.#.] _ %:: [ng:|%(ng%):] [a:|ạ:] [Bndry:|Con:|.#.] ;
```

```
"e:a = e^a | engi/ai"
e:a <=> _ [ .#. | %= ] ;
        [:Con|.#.] _ %:: [ng:|%(ng%):] [i:|u:] [Bndry:|Con:|.#.] ;
    except
        :Vow [Bndry:%>|:0]* t:n _ [ .#. | %= ] ;

"e:i = enga/ii"
e:i <=> [:Con|.#.] _ %:: [ng:|%(ng%):] [a:|ạ:] [Bndry:|:Con|.#.] ;

"g:0 = - | :"
g:0 <=> _ Bndry:* %-: ;
        [:Con|.#.] FVow: _ %%: %:: %(e%): Con: ;
        [:Con|.#.] FVow: _ [%@³:|%~:] %:: %(u%): Con: ;
        [:Con|.#.] FVow: _  %:: %(u%/i%): Con: ;
        [:Con|.#.] [Bndry:|:0]* FVow: %:: _ FVow: [Bndry:|Con:|.#.] ;
        [:Con|.#.] FVow: _ %:: FVow: [Bndry:|Con:|.#.] ;
        [:Con|.#.] FVow: _ %:: (%~:) %(ng%): FVow: [Bndry:|Con:|.#.] ;

"g:r = Uvular/VelarAgreement"
g:r <=> [r:|rr:] Bndry:* %-: Bndry:* _ ;

"g:k = word#"
g:k <=> _ [ .#. | %= ] ;

"gg:g = gte@(u)"
gg:g <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
         _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"gg:rr = Uvular/VelarAgreement"
gg:rr <=> [r:|rr:] Bndry:* %-: Bndry:* _ ;

"i:0 = --"
i:0 <=> _ Con:0 Bndry:* %-: %-: l:0 ;

"k̲:q = Uvular/VelarAgreement"
k̲:q <=> [r:|rr:] Bndry:* %-: Bndry:* _ ;

"l:0 = --"
l:0 => Vow:0 (Con:0) Bndry:* %-: %-: _ ;

"l:ll = te@ l/ll"
l:ll <=> t:0 (') e:0 (%°:) [Bndry:-AtSign:-%(r%):]* %@ : [Bndry:-AtSign:-%(r%):]* _ ;

"ll:l = lte@(u)"
```

```
ll:l <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
         _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"m:m̄ = @²m"
m:m̄ <=> Con t:0 e:0 [Bndry:-%°:-AtSign:]* %@²: Bndry:* _ ;

"n:n̄ = @¹n"
n:n̄ <=> Con t:0 e:0 (%°:) [Bndry:-AtSign:]* %@¹: [Bndry:-AtSign:]* _ ;

"ng:0 = :"
ng:0 <=> [:Con|.#.] [Bndry:|:0]* FVow: %:: _ FVow: [Bndry:|Con:|.#.] ;
    except
         Con t:0 e:0 [Bndry:-%°:-AtSign:]* %@²: [Bndry:-AtSign:]* _ ;

"ng:ng = @²ng"
ng:ng <=> Con t:0 e:0 [Bndry:-%°:-AtSign:]* %@²: [Bndry:-AtSign:]* _ ;

"q̲:k = Uvular/VelarAgreement"
q̲:k <=> g: Bndry:* %-: Bndry:* _ ;

"r:0 = - | % | :"
r:0 <=> _ Bndry:* %-: ;
        FVow _ [Bndry: - %*:]* %%:0 ;
        [:Con|.#.] FVow: _ %*: %%: %:: %(e%): Con: ;
        [:Con|.#.] FVow: _ [%@³:|%~:] %:: %(u%): Con: ;
        [:Con|.#.] FVow: _  %:: %(u%/i%): [Con:] ;
        [:Con|.#.] [Bndry:|:0]* FVow: %:: _ FVow: [Bndry:|Con:|.#.] ;
        [:Con|.#.] [Bndry:|:0]* FVow: _ %:: FVow: [Bndry:|Con:|.#.] ;
        [:Con|.#.] FVow: _ (%*:) %:: (%~:) %(ng%): FVow: [Bndry:|Con:|.#.] ;

"r̲:g = Uvular/VelarAgreement"
r̲:g <=> g: Bndry:* %-: Bndry:* _ ;

"r:q = word#"
r:q <=> _ (%*:) [ .#. | %= ] ;

"rr:0 = -"
rr:0 <=> _ Bndry:* %-: ;
    except
        _ Bndry:* %-: [k:|k̲:|q:|q̲:] Vow: ;

"r̲r̲:gg = Uvular/VelarAgreement"
r̲r̲:gg <=> g: Bndry:* %-: Bndry:* _ ;
```

```
"rr:r = rte@(u)"
rr:r <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
         _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"s:0 = te~s/c | (t)s/c"
s:0 <=> t:c (') e:0 Bndry:* _ ;
        %(t%):c Bndry:* _ ;

"ss:s = ste@(u)"
ss:s <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
         _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"t:0 = Cte@¹n/CCn̄ | Cte@²mngv/CCm̄ngvv | Cte@³(u)/Cu"
t:0 <=> Con _ e: (%°:) [Bndry:-AtSign:]* %@¹: ;
        Con _ e: (%°:) [Bndry:-AtSign:]* %@²: ;
        VlessFric:VoiceFric _ e: (%°:) [Bndry:-AtSign:]* %@³: ;
        _ (') e: (%°:) [Bndry:-AtSign:]* %@ : ;
        _ (') e: (%°:) [Bndry:-AtSign:-%(r%):]* %@ : ;

"t:c = tV--li/ci | te@y/c | te~s/c"
t:c <=> _ Vow:0 (Con:0) [Bndry:-%(u%):-AtSign:]* %-: %-: l:0 i ;
        _ (') e:0 [Bndry:-%(u%):-%(ng%):-AtSign:]* i ;
        _ (') e: (%°:) %:: %(u%/i%): m: ;
        _ (') e: (%°:) [Bndry:-%(u%):-AtSign:]* %@ : [Bndry:-%(u%):-AtSign:]* y: ;
        _ (') e: [Bndry:-%(u%):-AtSign:]* [s:|y:] ;

"t:l = Vte@²/l | te*@³(u)/l | te°@ gkng/l"
t:l <=> Vow _ (') e: %°: [Bndry:-AtSign:]* %@²: ;
        Vow _ (') e: %°: [Bndry:-AtSign:]* %@³: ;
        _ (') e: %°: [Bndry:-AtSign:]* %@ : ;

"t:n = WordFinalVte"
t:n <=> :Vow [Bndry:%>|:0]* _ e: [ .#. | %= ] ;

"t:q = te@ /q"
t:q <=> _ (') e: (%°:) [Bndry:-AtSign:]* %@ : ;

"t:s = Vte@²/s | Vt'e@³(u)/s'u | Cte@ gkng/s"
t:s <=> Vow _ (') e: [Bndry:-%°:-AtSign:]* %@²: ;
        Vow _ ' e: [Bndry:-%°:-AtSign:]* %@³: ;
        [Vow|VlessFric:VoiceFric|Nasal|Stop] _ (') e: [Bndry:-%°:-AtSign:]* %@ : ;

"t:y = Vte@³(u)/Vy"
t:y <=>
```

```
        Vow _ (') e:0 [Bndry:-%°:-AtSign:]* %@²: ;
        [Vow|Nasal|Stop] _ e: [Bndry:-%°:-AtSign:]* %@³: ;
        [Vow|VlessFric:VoiceFric|Nasal|Stop] _ (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"u:0 = --"
u:0 <=> _ Con:0 Bndry:* %-: %-: l:0 ;

"ugg:ug = ugte@(u)"
ugg:ug <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
           _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"urr:ur = urte@(u)"
urr:ur <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
           _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"v:vv = @²v"
v:vv <=> Con t:0 e:0 [Bndry:-%°:-AtSign:]* %@²: [Bndry:-AtSign:]* _ ;

"vv:v = vte@(u)"
vv:v <=> _ t:0 e:0 (%°:) [Bndry:-AtSign:]* %@³: ;
         _ t:s (') e:0 [Bndry:-%°:-AtSign:]* %@ : ;

"y:0 = te@y"
y:0 <=> t:c (') e:0 (%°:) [Bndry:-%(u%):-AtSign:]* %@ : [Bndry:-%(u%):-AtSign:]* _ ;
        t:c (') e:0 (%°:) [Bndry:-%(u%):-AtSign:]* _ ;
        %(t%):c Bndry:* _ ;

"y:s = Stop@y"
y:s <=> [VlessCon-t] (e:0) [Bndry:-AtSign:]* %@ : [Bndry:-AtSign:]* _ ;
        [VlessCon-t] (e:0) [Bndry:-AtSign:]* _ ;
    except
        t:c (') e:0 (%°:) [Bndry:-AtSign:]* %@ : [Bndry:-AtSign:]* _ ;
        t:c (') e:0 (%°:) [Bndry:-%(u%):-AtSign:]* _ ;
        %(t%):c Bndry:* _ ;

"(g):g = VVFinalBase"
%(g%):g <=> :Vow [:%>|:0]* :Vow [:%>|:0]* _ ;

"(g/t):g = VVFinalBase"
%(g%/t%):g <=> :Vow [:%>|:0]* :Vow [:%>|:0]* _ ;
    except
        :Con Bndry:* _ ;

"(g/t):t = CFinalBase"
```

```
%(g%/t%):t <=> :Con Bndry:* _ ;

"(t):t = CFinalBase"
%(t%):t <=> :Con Bndry:* _ ;
    except
        :Con Bndry:* _ Bndry:* [s:|y:] ;

"(t):c = (t)s/c"
%(t%):c <=> :Con Bndry:* _ Bndry:* [s:|y:] ;

"(u):u = notFVow"
%(u%):u <=> [Con:|e:] Bndry:* _ ;




"%:0 = percentWeak"
%%:0 <=> FVow: r: [Bndry:-%*:]* _ ;

"':' = ApostropheRule"
%':%' <=> .#. (Con) Vow Con e:0 [ Bndry: - %~: ]* _ ;
```

Appendix E

ESU.STRESS.TWOL FILE

```
Alphabet
    a c e g i k l m n p q r s t u v w y m̄ n̄
    vv ll ss gg rr ng ng ug ugg ur urr
    %[e%]
    '
    ạ ụ ġ
    á í ú ạ́ ụ́ â î û ậ ụ̂ é ê
    %[é%] %[ê%]
    %>
    %= ;

Sets
    B = %> ;

    Stop = p t c k q ;
    VoiceFric = v l s g r ug ur ġ ;
    VlessFric = vv ll ss gg rr ugg urr ;
    VoiceNasal = m n ng ;
    VlessNasal = m̄ n̄ ng ;

    VoiceCon = VoiceFric VoiceNasal ;
    VlessCon = VlessFric VlessNasal Stop ;

    Fric = VoiceFric VlessFric ;
    Nasal = VoiceNasal VlessNasal ;

    Con = Stop Fric Nasal w y ;

    FVow = a i u ạ ụ ;
    FVowStress = á í ú ạ́ ụ́ ;
    FVowHatted = â î û ậ ụ̂ ;
    AllFVow = FVow FVowStress FVowHatted ;
    Vow = FVow e %[e%] ;
    VowStress = FVowStress é %[é%] ;
```

```
    VowHatted = FVowHatted ê %[ê%] ;
    AllVow = Vow VowStress VowHatted ;

Rules


"a:â"
Vx:Vy <=> [.#.|:Con] B:* :Vow B:* :Con B:* _
~[ B:* :Vow :* | B:* :Con B:* :Con :* | B:* :Con B:* :' :* ] .#. ;
    except
        _ B:* (:Vow) B:* (:Con) B:* [.#.|:%=] ;
    where Vx in Vow
          Vy in VowHatted
    matched;

"a:á"
Vx:Vy <=> :Vow B:* _ ;
          [.#.|:Con] B:* _ B:* :Con B:* ([:Con|:']) B:* :Vow B:* :AllVow ;
          .#. (:Con) B:* _ B:* :Con B:* [:Con|:'] ;
          [.#.|:Con|:'] B:* :Vow B:* (:Con) B:* :Con B:* _ B:* :Con B:* :Con ;
          :Con B:* _ B:* :Con B:* :Con B:* :Vow B:* :Con B:* :AllVow ;
    except
        _ B:* (:Con) B:* [.#.|%-] ;
        [.#.|:Con] B:* :Vow B:* :Con B:* _
        ~[ B:* :Vow :* | B:* :Con B:* :Con :* | B:* :Con B:* :' :* ] .#. ;
    where Vx in Vow
          Vy in VowStress
    matched;
```

Appendix F

ESU.TWOL.XFST FILE

```
define Stop       [ p | t | c | k | q ];
define VoiceFric  [ v | l | s | g | r | ug | ur | ġ ];
define VlessFric  [ vv | ll | ss | gg | rr | ugg | urr ];
define VoiceNasal [ m | n | ng ];
define VlessNasal [ m̄ | n̄ | ng ];

define VoiceCon   [ VoiceFric | VoiceNasal ];
define VlessCon   [ VlessFric | VlessNasal | Stop ];

define Fric       [ VlessFric  | VoiceFric  ];
define Nasal      [ VlessNasal | VoiceNasal ];

define Con        [ Stop | Fric | Nasal | w | y ];

define FVow       [ a | i | u | ạ | ụ ];
define FVowStress [ á | í | ú | ạ́ | ụ́ ];
define FVowHatted [ â | î | û | ậ | ụ̂ ];
define AllFVow    [ FVow | FVowStress | FVowHatted ];
define Vow        [ FVow | e ];
define VowStress  [ FVowStress | é ];
define VowHatted  [ FVowHatted | ê ];
define AllVow     [ Vow | VowStress | VowHatted ];

define AutomaticDevoicing v -> vv,
                          l -> ll,
                          s -> ss,
                          g -> gg,
                          r -> rr,
                          ug -> ugg,
                          ur -> urr || [ Stop | VlessFric ] ">"* _  ,
                                       _ ">"* [ Stop | VlessFric ]            .o.
                          m -> m̄,
                          n -> n̄,
                          ng -> ng || [ Stop | VlessFric ] ">"* _   ;
```

```
define WeakInitialE [..] -> e   || .#. ["[e]"|"[é]"] Con _ ">"* Con    .o.
    ê -> e   || .#. ["[e]"|"[é]"] Con _ ">"* Con                      .o.
    [..] -> "'" || .#. ["[e]"|"[é]"] Con AllVow ">"+ Con _ ">"* AllVow   .o.
    [..] -> "'" || .#. ["[e]"|"[é]"] Con AllVow Con _ ">"+ AllVow        .o.
    ["[e]"|"[é]"] -> 0  || .#. _ Con AllVow ">"* Con                      ;

define urDeletion [ụ|ứ] r (->) u r         || g ">" _ l        .o.
    [ụ|ứ] r -> e ("'") "ur"                   || g ">" _ l               .o.
    r [ụ|ứ] r (->) r u                        || Vow Vow ">"+ _ l        .o.
    r [ụ|ứ] r -> "ur"                         || Vow Vow ">"+ _ l        .o.
    ê ">"+ r [ụ|ứ] r (->)  ">" r u            || _ l                     .o.
    ê ">"+ r [ụ|ứ] r -> e ">" ("'") "ur"      || _ l                     .o.
    [e|é] ">"+ r [ụ|ứ] r (->) ">" u "'" r     || _ l                     .o.
    [e|é] ">"+ r [ụ|ứ] r (->) e ">" ("'") "ur" || _ l                    .o.
    r [ụ|ứ] r (->) r u                        || VowHatted ">"+ _ l      .o.
    r [ụ|ứ] r -> "'" u r                      || VowHatted ">"+ _ l      .o.
    r [ụ|ứ] r -> ("'") "ur"                   || [Vow|VowStress] ">"+ _ l  ;

define DropHattedE ê -> 0 .o.
    [..] -> e || p _ ">"* p , t _ ">"* t , c _ ">"* c , k _ ">"* k , q _ ">"* q ,
        v _ ">"* v , l _ ">"* l , s _ ">"* s , g _ ">"* g , r _ ">"* r ,
        ug _ ">"* ug , ur _ ">"* ur ,
        vv _ ">"* vv , ll _ ">"* ll , ss _ ">"* ss, gg _ ">"* gg , rr _ ">"* rr ,
        ugg _ ">"* ugg , urr _ ">"* urr ,
        v _ ">"* vv , l _ ">"* ll , s _ ">"* ss, g _ ">"* gg , r _ ">"* rr ,
        ug _ ">"* ugg , ur _ ">"* urr ,
        m _ ">"* m , n _ ">"* n , ng _ ">"* ng , m̄ _ ">"* m̄ , n̄ _ ">"* n̄ ,
        ng _ ">"* ng ,
        w _ ">"* w , y _ ">"* y , c _ ">"* t , t _ ">"* c  ;

define TripleConsonant  rr   ->  0 || _ q ">"* Con               .o.
                        [..]  ->  e || Con _ ">"* [s|ss] ">"* Con  .o.
                        [..]  ->  e || Con ">"* t _ ">"* Con       .o.
                        [..] (->) e || Con ">"* Con _ ">"* Con     .o.
                        [..]  ->  e || Con _ ">"* Con ">"* Con      ;

define TripleVowel  AllFVow -> 0 ||
        [Con|.#.] ">"* AllFVow ">"* _ ">"* AllFVow ">"* [Con|.#.] ;
```

```
define ConnectiveMood  ạ -> 0     || _ ">" [a|á] \.#.   .o.
        ụ -> 0              || _ ">" [a|á] \.#.                          .o.
        [e|é] -> e ("'")    || [VowStress|VowHatted|Con] ">" ng _ [r|rr] .o.
        [e|é] -> "'" e ("'") || Vow ">" ng _ [r|rr]                       ;

define WordAdjustments ġ -> 0       || Vow _                          .o.
                       t (->) c     || \.#. _ [e|é] ">"* t ,
                                       \.#. _ [e|é] ">"* ss ">"* Stop  .o.
                       t [e|é] ">" rr (->) ">" q || _ Vow              .o.
                       â r [a|á] (->) a "'" a || ">" [q|k] _ q          .o.
                       [a|á] r [a|á] (->) a   || ">" [q|k] _ q         .o.
                       [a|â|á] -> e || ">" q _ ">"* [r|rr]              ;

define arDeletion
        "â" -> a "'" a,
        "î" -> i "'" i,
        "û" -> u "'" u  || _ ">"* Con ">"* "ạ" q ">"* .#.                    .o.
        "ạ" q -> 0 || _ ">"* .#.                                             .o.
        ["ạ"|"ą́"] [r|rr] -> 0 || Vow ">"* VowStress ">"* Con ">"* _ ">"* Con .o.
        "ạ" [r|rr] -> 0 || Con ">"* VowStress ">"* Con ">"* _ ">"* Con       .o.
        "â" -> a "'" á,
        "î" -> i "'" í,
        "û" -> u "'" ú  || _ ">"* Con ">"* ["ạ"|"ą́"] [r|rr] ">"* Con         .o.
        ["ạ"|"ą́"] [r|rr] -> 0 || Vow "'" VowStress ">"* Con ">"* _ ">"* Con  .o.
        a -> á "'",
        e -> é "'",
        i -> í "'",
        u -> ú "'" || _ ">"* Con ">"* "ą́" [r|rr] ">"* Con                     .o.
        "ą́" [r|rr] -> 0 || VowStress "'" ">"* Con ">"* _ ">"* Con            .o.
        "'" (->) 0 || VowStress _ ">"* Con                                    ;


define RemoveApostrophe "'" -> 0 ||
        [ .#. | Con ] ">"* AllVow ">"* Con ">"* _ ">"* AllVow ">"* AllVow ,
        VoiceCon ">"*  _ ">"* VoiceCon                                       ;

define AddApostrophe [..] -> "'" || VlessCon _ ">"* VoiceCon , n _ ">"* g   ;

define ToOrthography  vv -> v,
                      ll -> l,
                      ss -> s,
                      gg -> g,
                      rr -> r,
                      ugg -> ug,
```

```
                    urr -> ur || [ Stop | VlessFric ] ">"* _          ,
                              _ ">"* [ Stop ]                        .o.
                    m̄ -> m,
                    n̄ -> n,
                    ng -> ng || [ Stop | VlessFric ] ">"* _          .o.
                    rr -> r  || _ ">"* .#.                           .o.
                    ss -> s  || .#. ">"*  _                          .o.
                    [â|á|ạ|ậ|ạ́] -> a                                .o.
                    [ê|é] -> e                                       .o.
                    [î|í] -> i                                       .o.
                    [û|ú|ụ|ụ́|ự̂] -> u                               .o.
                    ["[e]"|"[é]"|"[ê]"] -> e                         .o.
                    ġ -> g                                           ;

define RemoveLigature m̄ (->) m     .o.
                      n̄ (->) n     .o.
                      ng (->) ng   .o.
                      ug (->) u g   .o.
                      ur (->) u r   .o.
                      ugg (->) u gg .o.
                      urr (->) u rr  ;

define FinalWordApostrophe Vow (->) "'" || Con _ .#. ;

regex AutomaticDevoicing   .o.
      WeakInitialE         .o.
      urDeletion           .o.
      DropHattedE          .o.
      TripleConsonant      .o.
      TripleVowel          .o.
      ConnectiveMood       .o.
      WordAdjustments      .o.
      arDeletion           .o.
      RemoveApostrophe     .o.
      AddApostrophe        .o.
      AutomaticDevoicing   .o.
      ToOrthography        .o.
      RemoveLigature       .o.
      FinalWordApostrophe   ;
```