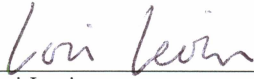



A LEXICAL TRANSDUCER FOR NORTH SLOPE INUPIAQ

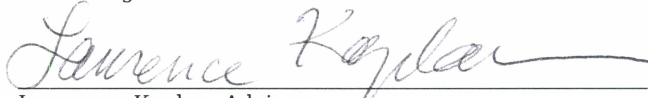
By

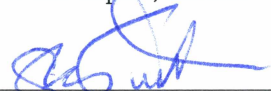
Aric R. Bills

RECOMMENDED:



Lori Levin

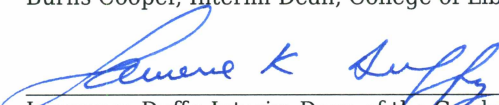

Anna Berge


Lawrence Kaplan, Advisor


Siri Tuttle, Chair, Linguistics Program

APPROVED:


Burns Cooper, Interim Dean, College of Liberal Arts


Lawrence Duffy, Interim Dean of the Graduate School


Date

A LEXICAL TRANSDUCER FOR NORTH SLOPE IÑUPIAQ

A
THESIS

Presented to the Faculty
of the University of Alaska Fairbanks

in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF ARTS

By

Aric R. Bills, B.A.

Fairbanks, Alaska

May 2011

Abstract

This thesis describes the creation and evaluation of software designed to analyze and generate North Slope Iñupiaq words. Given a complete Iñupiaq word as input, it attempts to identify the word's stem and suffixes, including the grammatical category and any inflectional information contained in the word. Given a stem and list of suffixes as input, it attempts to produce the corresponding Iñupiaq word, applying phonological processes as necessary. Innovations in the implementation of this software include Iñupiaq-specific formats for specifying lexical data, including a table-based format for specifying inflectional suffixes in paradigms; a treatment of phonologically-conditioned irregular allomorphy which leverages the pattern-recognition capabilities of the *xfst* programming language; and an idiom for composing morphographic rules together in *xfst* which captures the state of the software each time a new rule is added, maximizing feedback during software compilation and facilitating troubleshooting.

In testing, the software recognized 81.2% of all word tokens (78.3% of unique word types) and guessed at the morphology of an additional 16.8% of tokens (19.4% of types). Analyses of recognized words were largely accurate; a heuristic for identifying accurate parses is proposed. Most guesses were at least partly inaccurate. Improvements and applications are proposed.

Table of Contents

	Page
Signature Page	i
Title Page	ii
Abstract	iii
Table of Contents	iv
List of Figures	ix
List of Tables	xii
List of Other Materials	xii
List of Appendices	xii
List of Abbreviations	xiii
Acknowledgments	xv
Chapter 1 Introduction	1
Chapter 2 Background	3
2.1 Iñupiaq	3
2.1.1 Phonology and Orthography	5
2.1.1.1 Consonants	5
2.1.1.2 Vowels	6
2.1.2 Phonological Phenomena	7
2.1.2.1 Regressive Assimilation	7
2.1.2.2 Palatalization	9
2.1.2.3 Alternation of /i/ and [a]	11
2.1.2.3.1 Alternation Due to a Following Vowel	11
2.1.2.3.2 Alternation Due to Gemination of a Preceding Consonant	12
2.1.2.4 Prevention of Three-Vowel Clusters	14
2.1.2.5 Gemination	16
2.1.2.6 Suffix Attachment Patterns	18
2.1.2.7 /ti/-[n] Alternation	22
2.1.2.8 Allomorphy	23
2.1.3 Phonotactics	25
2.1.3.1 Consonant Clusters	25
2.1.3.2 Vowel Clusters	25
2.1.3.3 Morpheme and Word Structure	25
2.1.3.4 Distribution of Specific Phones	26
2.1.4 Morphology	27
2.1.4.1 Grammatical Categories	27
2.1.4.2 Categories of Word Formatives	28
2.1.4.3 Morphotactic Constraints	36
2.2 Computational Morphology	37
2.2.1 Dictionary Lookup	38

	Page	
2.2.2	Stemming	38
2.2.3	Finite-State Morphology	39
2.2.3.1	Definitions	40
2.2.3.2	Mathematical Properties of Finite-State Machines	42
2.2.3.3	Finite-State Approaches to Morphophonology	45
2.3	The Xerox Finite State Tools	50
2.3.1	Multicharacter Symbols	50
2.3.2	Lexicon Creation	51
2.3.3	Morphographemic Rules	53
2.3.4	Rule Triggers	54
2.3.5	Flag Diacritics and Filters	55
2.3.6	Dictionary Downtranslation	58
2.3.7	Stem Guessers	59
2.4	Langgård and Trosterud's Inupiaq Transducer	60
2.4.1	Morphographemics	60
2.4.2	Lexical Coverage	61
2.4.3	Lexical Architecture	61
2.4.3.1	Mechanics	61
2.4.3.2	Morphologically Motivated Features	62
2.4.3.3	Phonologically Motivated Features	64
2.4.4	File Structure and Compilation	66
Chapter 3	Implementation	67
3.1	Lexical Model	68
3.2	Formative Data Files	69
3.2.1	Stems	70
3.2.1.1	Syntax	70
3.2.1.2	Category Codes	72
3.2.1.3	Sources and Data Entry Methodology	76
3.2.2	Postbases	77
3.2.2.1	Syntax	77
3.2.2.2	Category Codes	80
3.2.2.3	Sources and Data Entry Methodology	83
3.2.3	Inflectional Endings	83
3.2.3.1	Category Codes and Metadata	84
3.2.3.2	Inflection Table Syntax	87
3.2.3.3	Accommodating Complex Paradigms Using Two-Dimensional Tables	92
3.2.3.3.1	Nominal Inflection	92
3.2.3.3.2	Intransitive Verbal Inflection	92
3.2.3.3.3	Transitive Verbal Inflection	93

	Page
3.2.3.3.4 Demonstrative Inflection	94
3.2.3.4 Sources and Data Entry Methodology	95
3.2.4 Enclitics	95
3.3 Definitions of Allomorph Conditioning Environments	96
3.3.1 Logic	98
3.3.2 Syntax	101
3.4 Formative Class Declarations	104
3.5 Declaration of Multicharacter Symbols	105
3.6 Conversion of Lexical Data to XML	108
3.6.1 Structure of the Resulting XML Document	108
3.6.2 Modifications Made to Data During Conversion to XML	110
3.7 Conversion of XML Data to XFST Formats	110
3.7.1 Resolution of Membership in More Than One Class	111
3.7.2 Resolution of Continuation to More Than One Class	112
3.7.3 Resolution of “Otherwise” Conditions	112
3.7.4 Outputting the Lexicon in lexc Format	113
3.7.5 Outputting Allomorph Conditioning Environment Data in xfst Format	113
3.8 Morphophonological Rules	114
3.8.1 Architecture and Idioms	114
3.8.2 The Rules	116
3.8.2.1 Loading the Lexicon; Eliminating Flag Diacritics	116
3.8.2.2 Global Convenience Definitions	117
3.8.2.3 Limiting Word Length	118
3.8.2.4 Eliminating Capital Letters from the Lexicon	119
3.8.2.5 Defining Stem Guessers	119
3.8.2.6 Filters for Irregular Inflected Forms	121
3.8.2.7 Reducing Two Consecutive Boundary Markers to One	122
3.8.2.8 Changing Noun-Stem-Final <i>-n</i> to <i>-ti</i>	122
3.8.2.9 Forming the Absolutive Dual	123
3.8.2.10 Handling Optional Gemination and Stems with “Historic Consonants”	124
3.8.2.11 Applying Conditioning Environment Filters	126
3.8.2.12 Deriving Demonstrative Stems	126
3.8.2.12.1 Demonstrative Adverb Stems	126
3.8.2.12.2 Demonstrative Pronoun Stems	128
3.8.2.13 “Weakening” /i/ Due to a Following Postbase	128
3.8.2.14 Gemination	129
3.8.2.15 Velarization of Stem-Final /q/	130
3.8.2.16 Uvularization of Enclitic-Initial /ɣ/	131
3.8.2.17 Vowel Lengthening	131

	Page
3.8.2.18 Handling Allomorphy Shorthand Notations	131
3.8.2.19 Suffix Attachment Patterns	133
3.8.2.19.1 Minus Pattern	133
3.8.2.19.2 Plus Pattern	133
3.8.2.19.3 Colon Pattern	134
3.8.2.19.4 Division Pattern	134
3.8.2.19.5 Plus Over Minus Pattern	134
3.8.2.19.6 Minus Over Plus Pattern	135
3.8.2.19.7 Equals Pattern	135
3.8.2.19.8 Tilde Pattern	136
3.8.2.20 Regressive Palatalization	137
3.8.2.21 Preventing /i/ from Becoming [a]	138
3.8.2.22 Breaking Up Three-Vowel Clusters	139
3.8.2.23 Optional Deletion of Laterals in Contemporative 1 Endings	139
3.8.2.24 Palatalization	140
3.8.2.25 Removing Morpheme Boundaries	141
3.8.2.26 Consonant Assimilation	141
3.8.2.27 Changing /i/ to [a] at the Beginning of Vowel Clusters	142
3.8.2.28 Standardizing Lower-Language Orthography	143
3.8.2.29 Converting Upper-Language Forms to a Standard Format	143
3.8.2.30 Saving the Compiled Transducers	144
Chapter 4 Evaluation	146
4.1 Corpus of North Slope Dialect Texts	146
4.2 Evaluation Procedures	147
4.3 Results	148
4.3.1 Recognition	148
4.3.2 Accuracy of Parses Not Involving Guessing	150
4.3.2.1 Characteristics of the Sample to Be Analyzed	150
4.3.2.2 Results	151
4.3.3 Analysis of Failures of the Non-Guessing Transducer	152
4.3.3.1 Stem-Related Failures	153
4.3.3.2 Postbase-Related Failures	154
4.3.3.3 Inflection-Related Failures	155
4.3.3.4 Enclitic-Related Failures	155
4.3.3.5 Failures Due to Phonological and Morphotactic Rules	155
4.3.3.6 Failures Due to Typographic Errors	156
4.3.3.7 Dealing with Causes of Failure	156
4.3.3.7.1 Postbases Not in Lexicon	156
4.3.3.7.2 Typos	156

	Page
4.3.3.7.3 Inadequate Phonological Rules	156
4.3.3.7.4 Stems Not in Lexicon	156
4.3.3.7.5 Unusual Inflectional Morphophonology	156
4.3.3.7.6 English Stems	157
4.3.3.7.7 Missing Phonological Rules	157
4.3.3.7.8 Inadequately Specified Postbases	157
4.3.3.7.9 Stem Variants Not in Lexicon	157
4.3.3.7.10 Inadequately Specified Inflectional Endings	157
4.3.3.7.11 Inadequate Morphotactics	157
4.3.3.7.12 Postbase Allomorphs Not in Lexicon	157
4.3.3.7.13 Reduced Forms Not in Lexicon	157
4.3.3.7.14 Incorrectly Specified Stems	157
4.3.3.7.15 Enclitic Not in Lexicon	157
4.3.3.7.16 Flaw in Postbase File Long Distance Dependency Logic . . .	158
4.3.3.7.17 Stem Not Specified as Transitive	158
4.3.3.7.18 Summary	158
4.3.4 Accuracy of Parses Involving Guessing	158
4.3.4.1 Characteristics of the sample to be analyzed	158
4.3.4.2 Results	159
Chapter 5 Conclusion	162
5.1 Contributions and Limitations	162
5.2 Possible Improvements to the Implementation of the Transducer	164
5.2.1 Specification of Inflection	164
5.2.2 A Preprocessor for xfst	164
5.2.3 Separate Permissive and Prescriptive Transducers	165
5.3 Possible Applications of the Transducer	165
5.3.1 Computer-Assisted Language Learning	165
5.3.2 Enhanced Dictionary Interface	166
5.3.3 Spell-Checker	166
5.3.3.1 Other Possible Applications	167
Appendices	168
Bibliography	186

List of Figures

	Page
2.1 Map of Iñupiaq-speaking areas in Alaska	4
2.2 Kaplan’s regressive assimilation rule	8
2.3 Fortescue’s sentential verbal suffix ordering rule	36
2.4 De Reuse’s “position-based” morphotactic model	37
2.5 An FSA that computes the regular language {can, cast, cost, man, mast, most}	40
2.6 An FSA modeling simplified Hawaiian phonotactics	41
2.7 An FST that computes present (non-3sg) and past forms of the words <i>run</i> , <i>read</i> , <i>rent</i> , and <i>risk</i>	41
2.8 AN FST that computes the regular relation corresponding to the rule $n \rightarrow \tilde{n} / \hat{i} _$	42
2.9 A portion of the forest-of-tries Finnish lexicon described in Koskenniemi (1983)	47
2.10 Formative classes defining a regular language containing some French infinitive verbs	52
2.11 Modified formative classes defining a regular relation	52
2.12 Accounting for a discontinuous dependency using formative classes and continuations	56
2.13 Formative classes which fail to account for discontinuous dependencies	56
2.14 Formative classes which account for discontinuous dependencies using flag diacritics	57
2.15 Simplified overview of the lexicon of Langgård and Trosterud’s Iñupiaq transducer	63
3.1 High-level schema of data and software used to produce the Iñupiaq transducer	67
3.2 Contents of the LongDistanceDependencies section of the stem file	71
3.3 Contents of the GrammaticalCategoryTags section of the stem file	71
3.4 Example entries from the Morphemes section of the stem file	72
3.5 Contents of the Categories section of the postbase file	78
3.6 Contents of the Continuations section of the postbase file	78
3.7 Contents of the LongDistanceDependencies section of the postbase file	79
3.8 Example entries from the Morphemes section of the postbase file	80
3.9 Post-inflectional suffixes implemented in special categories	81
3.10 Post-inflectional suffixes implemented as “hybrids”	82
3.11 Category definitions in inflection data file	85
3.12 Continuations defined in inflection data file	86
3.13 Long distance dependencies defined in inflection data file	86
3.14 Example inflectional table with no prefixes	89
3.15 Example inflectional table using rows with the -prefixes option	90
3.16 Example inflectional table using rows with the -prefixset option	91
3.17 Example inflectional table with “holes” (cells for which no suffix exists), marked by zeros	94
3.18 Inflectional tables implementing demonstrative inflection	95
3.19 Category and continuation definitions in enclitic data file	96
3.20 Regular expression model for filters not sensitive to the left edge of a word	98
3.21 Automaton without ignore operator, illustrating the expression [V C V]	99
3.22 Automaton with ignore operator, illustrating the expression [[V C V] / SymbolsToIgnore]	99
3.23 Regular expression for the “opposite” of the language expressed in Figure 3.20	101

	Page
3.24 Contents of the xfst section of the filter definition file	102
3.25 Contents of the Tcl section of the filter definition file	103
3.26 Example entries from the Patterns section of the filter definition file	104
3.27 Examples of formative class declarations and epsilon continuation definitions	106
3.28 Examples of multicharacter symbol declarations	107
3.29 Schematic of the structure of the XML file containing the lexicon	109
3.30 Treatment of formatives with multiple class memberships	111
3.31 A more optimal lexc treatment of the formatives in Figure 3.30	112
3.32 Treatment of formatives with multiple continuations	113
3.33 Code to load the lexicon and eliminate flag diacritics	117
3.34 Global convenience definitions	118
3.35 Code to limit word length	119
3.36 Code to limit word length	119
3.37 Convenience definitions for guessing stems	120
3.38 Rules for guessing noun and verb stems based on Iñupiaq phonotactics	121
3.39 Code to insert the stem-guessing rules into the transducer	121
3.40 Code to filter out unattested “regular” dual or plural members of a paradigm	122
3.41 Code to fix strings of two consecutive boundary markers	122
3.42 Code defining the /t̥i/-[n] alternation	123
3.43 Code for forming the absolutive dual	125
3.44 Code for handling optional gemination and “historic” consonants	125
3.45 Code for deriving stems from demonstrative adverbs	128
3.46 Code for deriving stems from demonstrative pronouns	129
3.47 Code for deriving stems from demonstrative pronouns	129
3.48 Code to implement gemination	130
3.49 Code to implement velarization of stem-final /q/	130
3.50 Code to implement uvularization of enclitic-initial /g/	131
3.51 Code to lengthen vowels	132
3.52 Code for handling bracket notation	132
3.53 Code for handling consonant over consonant notation	132
3.54 Code implementing the “minus” suffix attachment pattern	133
3.55 Code implementing the “plus” suffix attachment pattern	133
3.56 Code implementing the “colon” suffix attachment pattern	134
3.57 Code implementing the “division” suffix attachment pattern	135
3.58 Code implementing the “plus over minus” suffix attachment pattern	135
3.59 Code implementing the “minus over plus” suffix attachment pattern	135
3.60 Code implementing the “equals” suffix attachment pattern	136
3.61 Code implementing the “tilde” suffix attachment pattern	138
3.62 Code implementing the “tilde” suffix attachment pattern	138

	Page
3.63 Code to prevent /i/ from becoming [a]	138
3.64 Code to break up three-vowel clusters	139
3.65 Older code to break up three-vowel clusters (flawed)	139
3.66 Code to optionally delete laterals in certain contemporary 1 endings	140
3.67 Code to (optionally) palatalize alveolar consonants following /i/	140
3.68 Code to remove morpheme boundary symbols from the lower language of the transducer	141
3.69 Code implementing consonant assimilation	142
3.70 Code to change /i/ to [a] at the beginning of vowel clusters	143
3.71 Code to convert lower-language to standard Iñupiaq orthography	143
3.72 Code to standardize the transducer's upper language	143
3.73 Code to finalize the upper languages and save the guessing and non-guessing transducers	144
4.1 Lookup-strategy script used for evaluating the transducer	147
4.2 Guesser results: number of guesses plotted against word length in characters	149
4.3 Parses per recognized type: all recognized words vs. subset to be analyzed for accuracy	150
4.4 Parses per guessed type: all guessed words vs. subset to be analyzed for accuracy	159

List of Tables

	Page
2.1 Consonants of North Slope Iñupiaq	5
2.2 Cognate consonant clusters in North Slope Iñupiaq and Malimiut Iñupiaq	8
2.3 Graphemic reflexes of assimilation	9
2.4 MacLean’s suffix attachment patterns	19
2.5 “Post-inflectional” derivational suffixes	31
2.6 Closure of regular languages and relations under various operations	44
2.7 Two-level rule types	46
2.8 Two-level output compared to traditional morphological analysis	47
2.9 Flag diacritic types available in xfst	57
3.1 Stem category codes and associated metadata	73
3.2 Demonstrative stems realized as adverbs and pronouns	74
3.3 Suffix attachment symbols used in the transducer (see also Table 2.4)	79
3.4 Postbase category codes	81
3.5 Suffix attachment symbols in the lexical data files and in xfst/lexc files	111
3.6 Effects of the absolutive dual ending on the preceding stem	124
3.7 Examples of demonstratives with adverbial and pronominal citation forms and “stems”	127
3.8 Stem patterns and the “colon” rule operations which may affect them	135
4.1 Recognition results, not taking accuracy into account	148
4.2 Parses per word, by category	149
4.3 Comparison of parse statistics: all recognized words vs. subset to be analyzed for accuracy	150
4.4 Result of automatically selecting parses with fewest morpheme boundaries	153
4.5 Causes of non-guessing transducer failure	153
4.6 Comparison of parse statistics: all guessed words vs. subset to be analyzed for accuracy	159

List of Other Materials

CD-ROM with Transducer Source Code and Electronic Copy of Thesis	pocket
--	--------

List of Appendices

	Page
Appendix A Basics of xfst	168
Appendix B Grammar Tags Used in the Iñupiaq Transducer	171
Appendix C Files, Dependencies, Build Process, and Use of the Transducer	180

List of Abbreviations

1	first person
2	second person
3	third person non-reflexive (in dependent moods, indicates that the antecedent is not co-referent with the subject of the main verb)
3R	third person reflexive (indicates co-reference with the subject of the main verb; called 4th person in some grammars of Eskimo languages)
ABS	absolutive case
ANA	anaphoric
C	any consonant
COND	conditional (a dependent verb mood)
CONSEQ	consequential (a dependent verb mood)
CONTEMP1	contemporative 1 (a dependent verb mood)
COP	copula
DIR.OBJ	direct object
DU	dual (see also PL)
EXT	extended (of demonstratives; see also RESTR and NV)
FSA	finite-state automaton
FST	finite-state transducer
FUT	future
IMP	imperative
IND	indicative
IND.OBJ	indirect object
INT	interrogative
IPA	International Phonetic Alphabet
lit.	literally
lexc	a language in the Xerox Finite State Toolkit for defining lexicons (name derived from the phrase "Lexicon Compiler"; see also XFST, xfst)
MOD	modalis case
n.d.	no date
NEG	negative
NEGCONTEMP	negative contemporative (a dependent verb mood; also serves as negative counterpart of imperatives and intransitive optatives)
NLP	natural language processing
NV	not visible (of demonstratives; see also EXT and RESTR)
O	direct object
OPT	optative
P	possessor
PART	participial

PL	plural (≥ 3 ; see also <i>DU</i>)
PRS	“present” (not a true present tense)
PST	“past”
REAL	realized aspect
REL	relative case (marks possessors and definite subjects of transitive verbs)
REP	reportative
RESTR	restricted (of demonstratives; see also <i>ext</i> and <i>nv</i>)
SG	singular
SIM	similaris case
SIMUL1	simultaneitive 1 (a dependent verb mood)
Tcl	a general-purpose programming language created by former University of California Berkeley professor John Ousterhout; the name originally stood for “Tool Command Language”
TRM	terminals case
U+xxxx	Unicode code point xxxx (where <i>x</i> is a hexadecimal digit)
UNREAL	unrealized aspect
V	any vowel
VIA	vialis case
XFST	Xerox Finite State Toolkit (see also <i>xfst</i> , <i>lexc</i>)
xfst	a language from the Xerox Finite State Toolkit for defining finite-state machines (see also <i>XFST</i> , <i>lexc</i>)
XML	Extensible Markup Language

Acknowledgments

I wish to express my sincere gratitude to the many people who helped make this thesis possible.

First, I'm grateful to the National Science Foundation for funding this work (Award 0534217) and to Drs. Jaime Carbonell and Alon Lavie of Carnegie Mellon University, the principal investigator and co-principal investigator, respectively, for this grant.

It has been a privilege to have Dr. Lawrence Kaplan as my advisor. He has employed me, guided me in my study of Iñupiaq, patiently answered even the most elementary questions, and encouraged me throughout the entire process. His tremendous assistance in evaluating the transducer was invaluable. I could not have asked for a better advisor.

This thesis would not have happened without Dr. Lori Levin of Carnegie Mellon University, who has provided valuable guidance on the design and evaluation of the transducer from its inception. It was also thanks to her encouragement and assistance that we were able to present a paper on the transducer at SALTMiL 2010.

It is hard to adequately express my gratitude for everything Dr. Anna Berge has done for me. I was privileged to work for her for three years, during which time she taught me most of what I know about linguistic fieldwork and the Aleut language, and provided opportunities for me to learn about audio recording and processing (for which I am also indebted to talented sound engineer Ed Smith and to UAF professor Gary Holton). Along with Dr. Sabine Siekmann and Dr. Patrick Marlow, she supervised my (still incomplete) work digitizing Knut Bergsland's *Aleut Dictionary*. She has provided insightful comments on this thesis. For all this and much, much more, I offer my sincere thanks.

Per Langgård, Senior Advisor at Okaasileriffik (the Greenland Language Secretariat), and Dr. Trond Trosterud of the University of Tromsø have been most helpful. They provided me with the source code to their transducers, including an Iñupiaq transducer discussed in Chapter 2, which helped me learn `xfst` and `lexc`. They also provided valuable feedback on the SALTMiL 2010 paper, both before and after it was presented. My sincere thanks to them.

Dr. Edna MacLean has been generous with her wonderful Iñupiaq language materials, her expertise in the language, and her encouragement. This thesis could not have happened without her.

My undergraduate mentor, Dr. Deryle Lonsdale of Brigham Young University, has offered useful advice and encouragement throughout my graduate career, in addition to the wonderful tutelage and support he gave me as an undergraduate. I'm also grateful to Drs. Lane Steinagel, Mark Tanner, Willis Fails, Alan Manning, Cynthia Hallen, Alan Melby, Diane Strong-Krause, and Ray Graham, all of BYU, for helping me build a solid foundation in linguistics.

I'm grateful to Dr. Robert Frederking of Carnegie Mellon University, who provided useful advice and feedback during the creation of the transducer.

Graduate student Shinjae Yoo of Carnegie Mellon and undergraduates Ida Mayer, J. Eliot DeGolia, and Sai Venkateswaran of Carnegie Mellon and Paul Lundblad of the University of Pittsburgh created a corpus of North Slope Iñupiaq without which I could not have evaluated the transducer.

I wish to thank my Iñupiaq instructors, Dora Itta and Ronald Brower. Mr. Brower was also kind enough to let me do my linguistic internship in his beginning Iñupiaq class in Fall 2006; he also assisted Dr. Kaplan

in evaluating the output of the transducer, for which I am indebted to him.

I also express gratitude to my linguistics professors at UAF: Drs. Sabine Siekmann, Siri Tuttle, Gary Holton, and Patrick Marlow. Thanks are also due to the following individuals from the University of Washington: Dr. Sharon Hargus, who piqued my interest in the languages of North America and who encouraged me to attend UAF; Dr. Emily Bender, who, in the short time I was enrolled in her class, introduced me to LaTeX; and Drs. Hunter Hoffman and Todd Richards, who gave me opportunities to help them with their fascinating research.

Finally, I am grateful to my family for their support and patience throughout this process. My wife has made some tremendous sacrifices over the course of my graduate program, including following me to Alaska and waiting six years for me to graduate. Our children have been patient and encouraging. My parents have gone out of their way to help watch our children so that I could have a quiet environment in which to work (even while my mom finished a college degree of her own); they have also set wonderful examples for me, teaching me from an early age the importance of education. My father was also my first computer science teacher and laid the foundation for all my subsequent computational endeavors, including this thesis. And my parents-in-law have been kind and supportive in spite of everything I've put their daughter and grandchildren through. My deepest thanks to all of them.

Three of my grandparents have died during my time at UAF. All three were professional educators whose love of learning was contagious. All three earned master's degrees. I dedicate this thesis to their memory, and to my surviving grandmother, in recognition of their profound impact, direct and indirect, on my life.

Chapter 1 Introduction

The English-speaking world benefits tremendously from natural language processing (NLP) software. At present, NLP technologies for English are commonplace; these include, among others, spellcheckers, grammar checkers, document retrieval systems (notably including internet search engines), optical character recognition, speech synthesis and speech recognition, and machine translation. Many minority language communities would like to see at least some of these applications made available in their community's language (see for example Sarasola 2000; Uí Dhonnchadha 2003; Hussain 2004). Morphological analysis is fundamental to many NLP technologies, and can enhance many others; it stands to reason, then, that a first (or at least early) step toward NLP in a particular language is to develop a morphological analyzer for that language (Beesley 2004c; Sarasola 2000).

This document describes the development of a morphological analyzer/generator for North Slope Iñupiaq, an important native language of Alaska for which little (if any) NLP software is available. This analyzer/generator is implemented as a finite-state transducer, and could serve as the foundation for a spellchecker (Beesley and Karttunen 2003:451; Schulz and Mihov 2002) or as the lexical component of a part-of-speech disambiguator/tagger (Beesley and Karttunen 2003:454–455; Aduriz et al. 1995), and may be used to improve or facilitate the development of more complex NLP technologies for Iñupiaq, such as optical character recognition (Yoo 2008), document retrieval, or computer assisted language learning software.

A key goal in the development of the transducer has been to cover as much of the North Slope Iñupiaq lexicon as possible. The transducer's lexicon includes nearly all stems and enclitics from MacLean (n.d.a), as well as most postbases believed to be productive. The transducer also includes a component which will attempt to guess the stem of unrecognized words (see Section 3.8.2.5 on pages 119–121). However, there is currently no support for compound nouns such as *igliġutiksraŋa savaam* 'budget', and very little support for words of foreign origin which do not conform to native Iñupiaq phonotactics (e.g., *God*, English names, etc.).

The contributions of this thesis are twofold. First, language learners, researchers, and others who have occasion to analyze and/or synthesize Iñupiaq words will now have a tool to assist them in this process (though the tool will not, and should not, replace human judgment). Second, the unique implementation of the transducer may be of interest to other computational morphologists; in particular, it differs from most other lexical transducers of which I am aware in that its lexicon is not implemented directly in a finite-state language (such as Xerox's lexc format) but rather in a language-specific format which is then converted into lexc format for compilation. This approach allows lexical and inflectional data to be specified in more natural ways than would be possible, for example, using lexc directly (see Chapter 3, beginning on page 67; see also Bills et al. [2010]). The method used to compile morphographemic rules is also novel, to my knowledge; variables are defined at each stage in the compilation process, capturing the transducer's state each time a new rule is added. This allows XFST to provide more feedback during the compilation process, and makes debugging easier by allowing the developer to pinpoint the rule which causes output to deviate from what is expected, without having to recompile the transducer. The approach used for this thesis may be informative to those working on computational morphologies of polysynthetic languages,

languages with considerable phonologically conditioned irregular allomorphy, or languages with extensive inflection.

I've attempted to write this document in sufficient detail that a linguist with computational aptitude and a willingness to become familiar with the Xerox Finite State Tools (XFST) will be able to understand, modify, and extend the transducer's lexicon and rules. This document may also be useful to would-be computational morphologists who would like to see a complete example of how a transducer is implemented in XFST. Some sections of Chapter 2 may also be helpful to readers interested in Iñupiaq phonology or morphology but not necessarily in computational models of these.

The remainder of the document is organized as follows: Chapter 2 provides pertinent background information. This includes a discussion of the phonological phenomena and morphological characteristics of Iñupiaq which a computational morphology must take into account; a review of different approaches to computational morphology; an overview of the Xerox Finite State Tools, which are probably the most common framework for implementing computational morphologies today; and an examination of the proof-of-concept Iñupiaq transducer developed by Per Langgård and Trond Trosterud. Chapter 3 covers the implementation of the transducer: the conceptual model of the Iñupiaq lexicon used by the transducer; the specification of stems, postbases, inflectional endings, enclitics, and metadata used to construct the lexicon; the conversion of lexical data from project-specific formats into the lexc language; and the morphophonological rules used by the transducer. Chapter 4 provides an evaluation of the transducer's effectiveness in parsing Iñupiaq words drawn from published texts. Finally, Chapter 5 summarizes the major contributions of this thesis and proposes future improvements to the transducer and possible applications.

Chapter 2 Background

The organization of this chapter stems from the premise that in order to create a computational morphology engine for a particular language, one needs to know what linguistic facts are to be modeled, and one needs a framework in which to construct the model. Accordingly, Section 2.1 covers the most important aspects of Iñupiaq orthography, phonology, and morphology, and Section 2.2 discusses three major computational approaches to morphological analysis.

The only other computational model of Iñupiaq morphology of which I am aware is a lexical transducer created by Per Langgård and Trond Trosterud; this is reviewed in Section 2.4. To provide essential background information for this discussion and the one in Chapter 3, Section 2.3 explains some features of the Xerox Finite State Toolkit, which Kornai (1999:4) has called “the dominant finite state paradigm” for morphology.

2.1 Iñupiaq

Iñupiaq is an Eskimo language and part of the Inuit dialect continuum, which stretches from the Seward Peninsula in Alaska across northernmost Canada and into all inhabited areas of Greenland. Krauss (2007:408) reports that there were roughly 2100 speakers. In addition to a number of lexical differences, it is the most phonologically conservative variety of Inuit (Kaplan 1990:145): it retains Proto-Eskimo’s four-vowel system to a greater degree than any other Inuit dialect; it has a pair of retroflex consonants which, outside of the Alaskan branch of Inuit, are found only in the Natsilingmiutut subdialect of Inuktitut;¹ and it exhibits the most complex consonant clusters found in any variety of Inuit.

Iñupiaq itself can be subdivided into two main dialect groups, North Alaskan Iñupiaq and Seward Peninsula Inupiaq; each of these can be further subdivided into two dialects: North Alaskan Iñupiaq comprises the North Slope and Malimiut dialects, while Seward Peninsula Inupiaq consists of the Qawiaraq and Bering Strait dialects (Kaplan 1981c:7-8). North Alaskan Iñupiaq can be distinguished from Seward Peninsula Inupiaq by the presence of palatalization (see Section 2.1.2.2 on pages 9-11) and the absence Seward Peninsula Inupiaq’s unique “consonant weakening” processes, whose presence in those dialects is probably due to language contact with Central Alaskan Yup’ik (Kaplan 1981c:7; 2000).

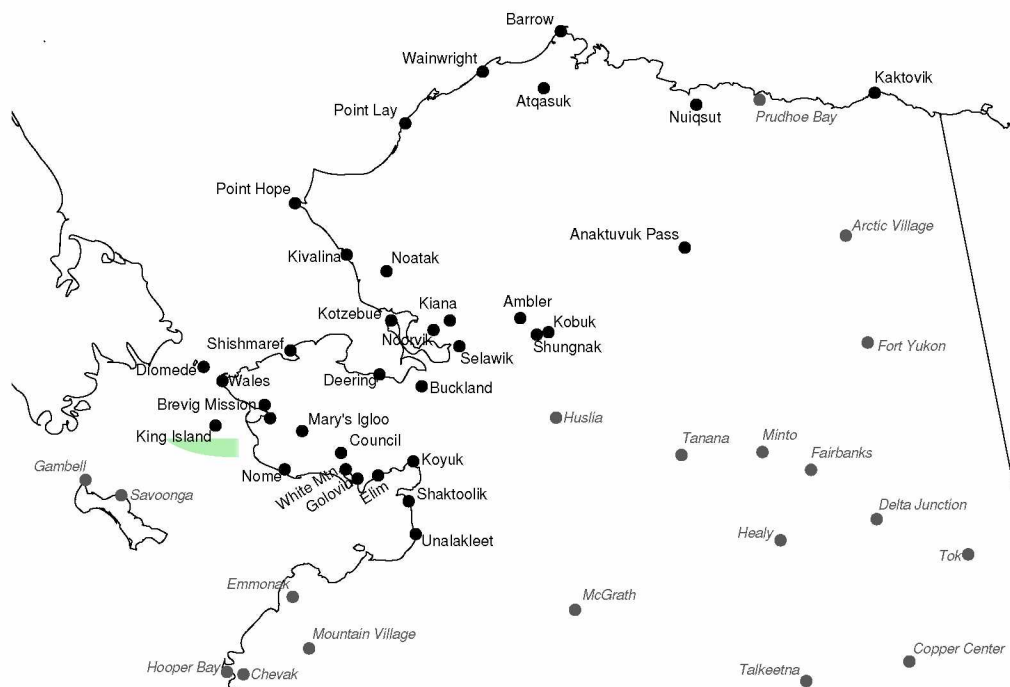
The present work deals with the North Slope dialect, which is spoken in the villages of Kivalina, Point Hope, Point Lay, Wainwright, Atqasuk, Barrow, Nuiqsut, Kaktovik, and Anaktuvuk Pass (MacLean [1986a:x]; see Figure 2.1 on the next page).^{2,3} Characteristics that set the North Slope dialect apart from

¹These consonants are remnants of Proto-Eskimo /*ð/. In all non-Iñupiaq dialects except Natsilingmiutut, this phoneme has merged with some other phoneme (Dorais 1990:39).

²The map in Figure 2.1 was created using the following GIS datasets from the Alaska Department of Natural Resources, Land Records Information Section, Anchorage, Alaska: “Simplified Alaska Coastline,” “Canada Coastline,” “Alaska DNR Russia Coastline,” and “Alaska Major Rivers,” all available from <http://www.asgdc.state.ak.us/>. Data was rendered using Quantum GIS 1.4.0 (<http://www.qgis.org/>) and Inkscape 0.46 (<http://inkscape.org/>). Coastline data was smoothed using Inkscape’s *simplify path* tool. Dialect areas, cities and villages were marked and labeled by hand and are approximate. Dialect areas are based on MacLean (1981:iv) and Kaplan (2000:90). Place names and locations are based on the “Populated Alaska Places” GIS dataset from the Alaska Department of Natural Resources, Land Records Information Section (see above).

³Uummarmiutun or Mackenzie Delta Iñupiaq, spoken in Northwest Territories, Canada, could also be considered a variety of North Slope Iñupiaq (Dorais 1990:46; Kaplan 1981c:78), but because virtually none of the sources on the phonology and lexicon of North Slope Iñupiaq cover this subdialect, I have excluded it from the present work.

the Malimiut dialect include a greater degree of consonant assimilation (or conversely, a smaller number of allowable consonant clusters), the absence of velar palatalization, and the presence of vowel clusters with different surface qualities (in Malimiut, all vowel clusters are reduced phonetically to long vowels), as well as a number of lexical differences. Although hard numbers are not available, perhaps roughly half of all Iñupiaq speakers speak the North Slope dialect (Lawrence Kaplan, personal communication, 29 March 2010).



2.1.1 Phonology and Orthography

2.1.1.1 Consonants

The consonants of North Slope Iñupiaq are presented in Table 2.1 (based on Kaplan [1981c:20-28; 2000:4-10] and MacLean [1986a:4-10]). In cases where the IPA symbol for a consonant differs from its representation in the standard Iñupiaq orthography, the phone is given in square brackets and its corresponding grapheme(s) in angle brackets. Consonants not considered phonemes in their own right are italicized and grouped in a shaded box with the primary allomorph of the phoneme. [tʃ] and [s] represent a special case, and will be discussed on page 11 in Section 2.1.2.2.⁴

TABLE 2.1: Consonants of North Slope Iñupiaq

	Labial	Alveolar	Palatal	Retroflex	Velar	Uvular	Glottal
Stops	p	t	[tʃ] (ch, t)		k	q	[ʔ] (ʔ)
Voiceless fricatives	[f] (v)	s		[ʂ] (sr)	[x] (kh, k)	[χ] (qh, q, h)	h
Voiced fricatives	v		[j] (y)	[z] (r)	[ɣ] (g)	[ʁ] (ġ)	
Nasals	m	n	[ɲ] (ñ)		ŋ	[ŋ] (ġ)	
Voiceless laterals		[ɬ] (t)	[ʎ] (t)				
Voiced laterals		l	[ʎ] (l)				

In the standard Iñupiaq orthography, a number of consonants represent more than one sound. ⟨q⟩ and ⟨k⟩ represent fricatives when adjacent to fricatives or laterals, and stops otherwise. ⟨v⟩ represents a voiceless fricative when adjacent to other voiceless fricatives, and a voiced fricative elsewhere. In some dialects, ⟨ġ⟩ is a nasal when adjacent to another nasal, and a voiced fricative elsewhere. ⟨t⟩ represents [tʃ] when preceded by a palatalizing /i/ (see Section 2.1.1.2) and [t] elsewhere. Unlike other consonants mentioned in this paragraph, it is not always possible to determine from the standard spelling of a word which pronunciation of ⟨t⟩ applies: in *tikitpa* ‘did he/she arrive?’ the second ⟨t⟩ represents [tʃ]; in *makitpa* ‘did he/she stand up?’ the ⟨t⟩ represents [t]. Iñupiaq pedagogical materials sometimes add a dot below the ⟨t⟩ to indicate the pronunciation [tʃ] (e.g., *tikitpa*) and this convention will also be followed in this document in cases where this distinction is important to the discussion.

⟨h⟩ represents [h] at the beginning of a handful of interjections (for example, *hauk* ‘I’m tired!’). Following a consonant, ⟨h⟩ is pronounced [χ] or [h] depending on the speaker (Kaplan 1981c:20, 22).

⟨h⟩, ⟨k⟩, ⟨q⟩, ⟨s⟩, and ⟨r⟩ also do double duty in that they occur both on their own and as part of one

⁴The complementary distribution of these pairs are documented as follows: [v] and [f], Kaplan (1981c:25); [ʁ] and [ŋ], Kaplan (1981c:25-26); [n] and [ɲ], [l] and [ʎ], [t] and [ʎ], Kaplan (1981c:90). In order for these last three pairs to be considered in complementary distribution, the reader must accept the existence of two underlying phonemes corresponding to surface [i], a claim which will be addressed in section 2.1.1.2 on the following page.

Note that while Kaplan acknowledges the complementary distribution of [tʃ] and [s], he prefers to treat the two sounds as separate phonemes except in the case of palatalization, in which case both are allophones of /t/ (Kaplan 1981c:86, 94-96, 176-179).

or more digraphs: ⟨kh⟩, ⟨qh⟩, ⟨ch⟩, and ⟨sr⟩. ⟨c⟩ has the distinction of being the only letter in the Iñupiaq orthography which only occurs as part of a digraph.

Iñupiaq makes a phonetic distinction between short and long consonants. Apart from consonants [f], [ʃ], [x], [χ], [h], and [ɲ], all of the consonants in Table 2.1 have a long counterpart (Kaplan 1981c:32); in most cases, this is simply the geminated form of the corresponding short consonant, and is written by doubling the appropriate grapheme (Kaplan 2000:93). The long counterpart of ⟨ch⟩, [tʃ:], is represented in the orthography as ⟨tch⟩; ⟨s⟩ also geminates to ⟨tch⟩. In some lexically conditioned cases, [g] geminates to ⟨kk⟩ [k:], [ɬ] geminates to ⟨qq⟩ [q:], and [y] geminates to ⟨tch⟩ [tʃ:] (MacLean [1986a:27]; see Section 2.1.2.5, particularly pages 17–18). Although there is a productive gemination process in Iñupiaq, not all long consonants result from this process; many result from regressive assimilation of a consonant cluster or simply represent the chance occurrence of two identical phonemes in a row (Kaplan 1981c:221).

A non-phonetic (and arguably non-phonological) distinction is made between so-called “weak” and “strong” /q/. The distinction is relevant only for a handful of suffixes which attach to noun stems; these suffixes delete stem-final “weak” /q/ but preserve other stem-final consonants (Kaplan 1981c:230–231). One such suffix is the ablative singular ending *-miñ*. This suffix triggers deletion of the “weak” /q/ at the end of *qimmiq* ‘dog’, yielding *qimmimiñ* ‘from the dog’; by contrast, the suffix does not delete the “strong” /q/ at the end of *aviñḡaq* ‘lemming’, instead leaving it to be partially assimilated to ⟨ḡ⟩ by the following nasal: *aviñḡaqmiñ* ‘from the lemming’ (examples from MacLean 1986a:116–117). The distinction between weak /q/ and strong /q/ is purely abstract; there is no phonetic difference between the two (Kaplan 1981c:229, 231).⁵ The disposition of a particular /q/ is only partially determined by the preceding phonetic environment (MacLean [1986a:76–77]; phonetic constraints on weak and strong /q/ will be discussed on pages 26–27 in Section 2.1.3.4). Throughout this document, I will use the notation /q̄/ to denote weak /q/ and /q̇/ to denote strong /q/. (For more information on the suffixes for which this distinction is pertinent, see the discussion of the ‘÷’ pattern in Section 2.1.2.6, especially page 21.)

2.1.1.2 Vowels

On the surface, North Slope Iñupiaq has a system of three short vowels (Kaplan 1981c:31), represented orthographically as ⟨a⟩, ⟨i⟩, and ⟨u⟩. Iñupiaq allows all possible two-letter combinations of these three vowels, including long vowels. The exact phonetic realization of short vowels, long vowels, and vowel clusters can vary considerably depending on adjacent phones (Kaplan 1981a:6; 1981c:32, 164) and from dialect to dialect (Kaplan 1981c:32, 164–165).⁶

Although North Slope Iñupiaq (like most of the Inuit continuum) has a three-vowel system, Proto-

⁵However, whatever phenomenon in Proto-Eskimo led to the distinction between strong and weak [q] probably did involve a phonetic distinction (Kaplan 1981c:233).

⁶From the descriptions in Kaplan (1981a:6) and MacLean (1986a:11), ⟨a⟩, ⟨i⟩, and ⟨u⟩ would correspond very roughly to prototypical phones [ʌ], [i], and [u], respectively. I am not aware of published studies on the phonetics of North Slope Iñupiaq vowels, but from informal observation it is clear that for any vowel phoneme, the first and second formants vary considerably depending on adjacent phones (Taff et al. [2001] observed in Pribilof Island Aleut, a related language with a similar three-vowel system, that vowel quality is often significantly affected by the place of articulation of adjacent consonants; uvulars triggered the largest changes in vowel quality, which seems to be the case in Iñupiaq as well). In the absence of hard phonetic data on North Slope Iñupiaq vowels, I will use the notations [a], [i], and [u] to represent surface vowels corresponding to ⟨a⟩, ⟨i⟩, and ⟨u⟩, respectively, even when the actual vowels produced by a native speaker might be more accurately represented by other IPA symbols. While this notation will be phonetically unreliable, it will make clear whether or not an underlying vowel is realized faithfully, that is, as an allophone that a native speaker would recognize as an instance of the underlying vowel.

Eskimo had four vowels, as do today’s Yupik languages (Kaplan 1981c:76; Jacobson 1984:8). The fourth vowel of Proto-Eskimo is reconstructed as /*ə/ and corresponds to Yupik (e). Phonetically, the North Slope Iñupiaq reflex of Proto-Eskimo /*ə/ is indistinguishable from the reflex of Proto-Eskimo /*i/, and both are written (i). Phonologically, however, artifacts of the distinction persist, as evidenced by the “split personality” of (i):

- Instances of (i) which derive from /*i/ (almost) never precede alveolar consonants other than /s/, or consonant clusters containing alveolars other than /s/; those which derive from /*ə/ never precede palatals other than /j/, or consonant clusters containing palatals other than /j/ (Kaplan [1981c:91]; see Section 2.1.2.2 on page 9). For example, the (i) in *tupiġ* ‘tent’ (which comes from /*ə/) may not precede palatals, while the (i) in *savik* ‘knife’ (from /*i/) may not precede alveolars; thus, *tupiligaaqtuaq* ‘he/she took along a tent’ contains ⟨l⟩ while *saviligaaqtuaq* ‘he/she took along a knife’ contains ⟨l⟩.
- Only the instances of (i) which reconstruct to /*ə/ undergo a change in vowel quality when the following phoneme is a vowel (Kaplan [1981c:91, 118–121, 125–126]; see page 12) or when the preceding consonant has undergone gemination and the following suffix begins with a velar or uvular (Kaplan [1981c:121–124]; see pages 13–14). For example, the derivational suffix *-aluk* ‘old; worn-out’ and the dual marker *-k* trigger changes in vowel quality in *tupaaluk* ‘old tent’ and *tuppak* ‘two tents’ but not in *savialuk* ‘old knife’ or *savvik* ‘two knives’ (data from MacLean [n.d.a, 1986a:78]).
- Certain suffixes trigger deletion of /*ə/-derived (i) (but not /*i/-derived (i)) in stems to which they attach.⁷ One such suffix is the plural marker *-it̚*: the (i) of *tupiġ* is deleted in the plural (*tupqit̚*) while the (i) of *savik* remains (*saviiġ*) (data from MacLean 1986a:79).

Thus, (i) comes in two distinct flavors. The one that may precede palatals and not alveolars is generally cognate with Yupik (i) and derived from Proto-Eskimo /*i/, and is sometimes called “strong i” in the literature (see for example MacLean [1986a:19]; Dorais [1990:48–49]); the other Iñupiaq (i), so-called “weak i,” is generally cognate with Yupik (e) and derived from Proto-Eskimo /*ə/ (Kaplan 1981c:76–77, 83–84, but see Kaplan 1981b). In keeping with convention (Kaplan 1981c, 2000), I will represent the North Slope Iñupiaq derivative of Proto-Eskimo /*ə/ as *ī*; breaking from the same convention, I will represent the derivative of /*i/ as *i̇*, in order to remove some ambiguity from the symbol *i*, which I will use when the distinction is unimportant.

2.1.2 Phonological Phenomena

2.1.2.1 Regressive Assimilation

North Slope Iñupiaq has a productive regressive assimilation process. Although alternations resulting from this process can only be observed synchronically at morpheme boundaries, the same assimilation process has also applied diachronically within morphemes (Kaplan 1981c:42–43), as can be seen by comparing North Slope words with cognates in other Alaskan Iñupiaq dialects (see Table 2.2 on the following page).

⁷Deletion is also contingent on the phonetic environment of the (i); see Section 2.1.2.6, particularly the discussion of the ‘:’ pattern on pages 20–21, for more details.

TABLE 2.2: Cognate consonant clusters in North Slope Iñupiaq and Malimiut Iñupiaq

North Slope	Malimiut	English gloss
<i>qimmiq</i>	<i>qipmiq</i>	'dog'
<i>imnaiq</i>	<i>ipnaiq</i>	'sheep'
<i>navlu</i>	<i>naplu</i>	'knee joint'
<i>qarraq</i>	<i>qatraq</i>	'an echo'
<i>pallik-</i>	<i>patlik-</i>	'be by the edge of something'
<i>pavri-</i>	<i>papri-</i>	'bother or pester someone'

The rule for North Slope regressive assimilation given by Kaplan (1981a:41) is reproduced in Figure 2.2. This rule stipulates that, given two adjacent consonants C_1 and C_2 , C_1 will adapt as necessary to agree with C_2 in voicing and continuancy; if C_1 is not a back consonant, it will additionally agree in terms of nasality, and if it is a coronal (in other words, alveolar, palatal, or retroflex), it will additionally agree in terms of laterality. (For Kaplan's analysis of the distinctive features of Iñupiaq consonants, see Kaplan [1981c:29]).

$$\left[\left\langle \begin{array}{c} C \\ \text{-back} \\ \langle +\text{coronal} \rangle \end{array} \right\rangle \right] \rightarrow \left[\left\langle \begin{array}{c} \alpha \text{ continuant} \\ \beta \text{ voice} \\ \gamma \text{ nasal} \\ \langle \delta \text{ lateral} \rangle \end{array} \right\rangle \right] / \text{---} \left[\begin{array}{c} \alpha \text{ continuant} \\ \beta \text{ voice} \\ \gamma \text{ nasal} \\ \delta \text{ lateral} \end{array} \right]$$

FIGURE 2.2: Kaplan's regressive assimilation rule

The rule given in Figure 2.2 must be qualified by three considerations. First, in some subdialects of North Slope Iñupiaq, even back consonants must agree with the following consonant in nasality (Kaplan 1981c:39; MacLean 1986a:28). Second, according to MacLean (1986a:15–17, 28), some dialects allow nasals to remain unchanged when followed by /l/. Finally, sequences /ts/ and /ty/ do not undergo regressive assimilation, instead becoming [tʃ:] (MacLean 1986a:29; 1981:83).

The graphemic correlate of assimilation is slightly simpler than its phonological counterpart due to the fact that a single letter in the Iñupiaq alphabet often represents two homorganic consonants. For example, although the phoneme /q/ assimilates to [χ] before a voiceless continuant, it is written (q) in both cases, and thus no graphemic rule is necessary. Table 2.3 on the following page gives examples of graphemes which may change form due to assimilation.

TABLE 2.3: Graphemic reflexes of assimilation

Underlying	Surface	Environment	Example
(k)	(g)	voiced consonant	<i>savik</i> ‘knife’ + <i>-mik</i> MOD.SG <i>savigmik</i> ‘with a knife’
(k)	(ŋ) / (g)	nasal	<i>savik</i> ‘knife’ + <i>-mik</i> MOD.SG <i>saviŋmik</i> / <i>savigmik</i> ‘with a knife’
(m)	(v) / (m)	(l)	<i>iglum</i> ‘house (REL.SG)’ + <i>=lu</i> ‘and’ <i>igluvlu</i> / <i>iglumlu</i> ‘and (of) the house’
(n)	(l) / (n)	(l)	<i>tiŋŋun</i> ‘airplane’ + <i>=lu</i> ‘and’ <i>tiŋŋullu</i> / <i>tiŋŋunlu</i> ‘and the airplane’
(q)	(ġ)	voiced consonant	<i>miquq-</i> ‘sew’ + <i>-niaq-</i> FUT + <i>-tuq</i> IND.PRS.3SG <i>miquġniaqtuq</i> ‘he/she will sew’
(t)	(l)	(l)	<i>annuġaat</i> ‘clothing (PL)’ + <i>=lu</i> ‘and’ <i>annuġaallu</i> ‘and the clothing’
(t)	(t̪)	(t̪)	<i>aqpat-</i> ‘run (of human)’ + <i>-t̪uni</i> CONTEMP1.REAL.3SG <i>aqpat̪uni</i> ‘he/she running, ...’
(t)	(n)	any nasal	<i>aqpat-</i> ‘run’ + <i>-niaq-</i> FUT + <i>-tuq</i> IND.PRS.3SG <i>aqpanniaqtuq</i> ‘he/she will run’
(t)	(r)	(v, r, g) (but see next line)	<i>it-</i> ‘be, exist’ + <i>-vik</i> ‘place/time/source’ <i>irvik</i> ‘usual place; place to be’
(t, ch)	(y)	(g)	<i>qimmit/qimmich</i> ‘dogs’ + <i>=gguuq</i> ‘it is said that’ <i>qimmiyguuq</i> ‘it is said that dogs ...’

2.1.2.2 Palatalization

Palatalization is a distinguishing characteristic of North Alaskan Iñupiaq. Most Inuit dialects exhibit no palatalization at all, and when non-Alaskan dialects⁸ do exhibit palatalization (or more accurately, assibilation), the only affected phoneme is /t/ (Compton and Drescher 2008; Kaplan 1981c:78), whereas in North Slope Iñupiaq, phonemes /t/, /l/, /ʎ/, and /n/ are all subject to palatalization; that is, in the environment *î(C)___*, these alveolar phonemes are realized as their phonemically (if not phonetically) palatal allophones: /t/ becomes [t̪] or [s], /l/ becomes [ʎ], /ʎ/ becomes [ʎ̪], and /n/ becomes [ɲ]. (Malimiut Iñupiaq exhibits an even greater range of palatalization phenomena; see Kaplan [1981c:96–103].)

Examples (1)–(5) below illustrate the alternation between alveolar and palatal consonants. Underlying alveolars are realized as palatal consonants following /î/ (‘a’ examples); following /i/ (‘b’ examples) or any other vowel (‘c’ examples), they are realized as alveolar.

⁸An exception is the Mackenzie Delta or Uummarmiutun dialect, which is the result of an eastern migration of Alaskan Inuit around the turn of the 20th century (Kaplan 1981c:78; Dorais 1990:46).

- Ex. (1) a. *natchiġlu*
natchiq=lu
 seal=and
 ‘and the seal’
 b. *aiviġlu*
aiviq=lu
 walrus=and
 ‘and the walrus’
 c. *aviŋŋaġlu*
aviŋŋaq=lu
 lemming=and
 ‘and the lemming’
- Ex. (2) a. *aŋutigikłuni*
aŋutigik-łuni
 be.handsome-CONTEMP1.REAL.3SG
 ‘he being handsome, ...’
 b. *naŋiłłuni*
naŋit-łuni
 be.sick-CONTEMP1.REAL.3SG
 ‘he/she being sick, ...’
 c. *piñaqłuni*
piñaq-łuni
 be.generous-CONTEMP1.REAL.3SG
 ‘he/she being generous, ...’
- Ex. (3) a. *niġiñiaqtuq*
niġi-niaq-tuq
 eat-FUT-IND.PRS.3SG
 ‘he/she/it will eat’
 b. *iqaġiñiaqtuq*
iqaġi-niaq-tuq
 wash.hands/face-FUT-IND.PRS.3SG
 ‘he/she will wash his/her hands
 and/or face’
 c. *iglaŋaniaqtuq*
iglaŋa-niaq-tuq
 smile-FUT-IND.3SG
 ‘he/she will smile’
- Ex. (4) a. *kaviqsuq*
kaviq-tuq
 be.red-IND.PRS.3SG
 ‘it is red’
 b. *qatiqtuq*
qatiq-tuq
 be.white-IND.PRS.3SG
 ‘it is white’
 c. *maŋaqtuq*
maŋaq-tuq
 be.black-IND.PRS.3SG
 ‘it is black’
- Ex. (5) a. *aullarriṭ / aullarriċ*
aullarri-t
 leader-ABS.PL
 ‘leaders’
 b. *iñuunniaqtit*
iñuunniaqti-t
 physician-ABS.PL
 ‘physicians’
 c. *Kuuvaŋmiut*
Kuuvaŋmiu-t
 inhabitant.of.Kobuk.Valley-ABS.PL
 ‘inhabitants of the Kobuk Valley’

A peculiarity surrounding palatalization in North Slope Iñupiaq is that not all palatal phonemes share the same distribution or behavior. Laterals and nasals are straightforward enough; palatal laterals and

nasals only occur following /i/ (Kaplan 1981c:90).⁹ The other palatal consonants, [j] and [tʃ], have a much wider distribution, as evidenced by the following words: *ayuktaq* ‘ball’, *uyaġak* ‘stone, rock’, *katchi* ‘wall’, *igutchaq* ‘bumblebee’. [j] simply doesn’t behave like other palatals with respect to the palatalization process; it is a phoneme in its own right, derived from Proto-Eskimo /*j/ (Dorais 1990:48). [tʃ] is more complex; it does alternate with [t], as in example (5), but only when /i/ precedes; in other environments there is no reason to analyze [tʃ] as underlying /t/ (Kaplan 1981c:90).

The situation of [s] also deserves some attention. Although phonetically alveolar, [s] behaves like a palatal in that it alternates with [t] following /i/. In fact, [tʃ] and [s] are in complementary distribution; [s] cannot be long, cannot begin a consonant cluster, and cannot occur morpheme-finally; [tʃ] cannot occur word-initially, cannot be short except morpheme-finally, and cannot end a consonant cluster other than [tʃ:] (Kaplan 1981c:86). Without any loss of descriptive adequacy, one could analyze non-alternating [s] as the pre-vocalic allophone of /tʃ/, but Kaplan finds this analysis unsatisfactory because short [tʃ] never occurs in surface forms except as an allophone of /t/. Alternatively, one could analyze [tʃ] as an allophone of /s/; this would imply that /t/ palatalizes to [s] in all cases and then becomes [tʃ] when no vowel follows. Kaplan finds this implausible because it would involve changing a stop to a fricative to a stop. Because both single-phoneme analyses present explanatory drawbacks, Kaplan prefers to analyze [tʃ] and [s] as separate phonemes when they do not alternate with /t/; when they do alternate with /t/, he derives [s] from [tʃ] in the environment X__V where X is any segment except [tʃ]; this rule leaves [tʃ:] unchanged. In this work I will also treat non-palatalized [tʃ] and [s] as separate phonemes, but this should be seen more as a matter of convenience than as a theoretical claim on my part: the phonological rules in MacLean (1986a) are also (implicitly) written from a separate-phoneme point of view, and adopting this view facilitates computational adaptation of these rules.

2.1.2.3 Alternation of /i/ and [a]

Two environments cause an underlying /i/ to change to [a]. The first is a following vowel; the second is gemination of the preceding consonant. These will be considered in turn.

2.1.2.3.1 ALTERNATION DUE TO A FOLLOWING VOWEL With very few exceptions, North Slope Iñupiaq /i/ is realized as [a] when another vowel follows (Kaplan 1981c:118-121). Examples of this phenomenon are given below (data from MacLean [n.d.a]).

- | | | | |
|------------|--|----|---|
| Ex. (6) a. | <i>niq̄at̄chiāġniaqtuṅa</i>
<i>niq̄i-at̄chiaq-niaq-tuṅa</i>
meat-ask.other.household.for-FUT-IND.PRS.3SG
‘I will go ask (another household) for meat’ | c. | <i>atiḡaukkaqtut</i>
<i>atiḡi-ukkaq-tut</i>
parka-possess.many-IND.PRS.3PL
‘they have many parkas’ |
| b. | <i>anuḡaiqsuq</i>
<i>anuḡi-îq-tuq</i>
wind-lack-IND.PRS.3SG
‘(the weather) is calm, lit. lacks wind’ | | |

⁹I am aware of two minor exceptions: the interjection *ñiaq* ‘don’t do that!’ (Kaplan 1981c:32-33) and the verb stem *lahlak-* ‘to bark ferociously and be about to fight (of dog)’ (MacLean n.d.a).

Suffixes *-anik-* ‘previously, already’ and *-aqsi-* ‘about to; beginning to’ represent an exception to this rule; they cause a preceding /i/ to be realized as [i] (Kaplan 1981c:120–121), as illustrated in (7) below (data from MacLean [n.d.a]); this may be due to an underlying consonant at the beginning of these postbases (Lawrence Kaplan, personal communication, 21 March 2011).

Ex. (7) a.	\downarrow <i>supianiktaa</i> <i>supi-anik-taa</i> blow.out-already-IND.PRS.3SG>3SGO ‘he/she already blew it out’	b.	\downarrow <i>itiaqsiruq</i> <i>itiq-aqsi-ruq</i> wake.up-begin-IND.PRS.3SG ‘he/she began to wake up’
------------	---	----	---

/i/ may begin a stem but not a suffix (Kaplan 1981c:105, 119). Consequently, there are very few situations where /i/ could follow another vowel; Kaplan identifies two. The first is the demonstrative *tainna* ‘in that way or manner’, formed from Iñupiaq’s only prefix, *ta(t)-* ‘that specific person, thing, or concept; that person or thing closer to the listener than the speaker’ and the stem *inna* ‘like this’, whose initial vowel is /i/, as evidenced by the alveolar consonants that follow. The other example, also mentioned by MacLean (1986a:118), is the suffix pair *-miit-/niit-* ‘be located in, at, on ____’ (*-miit-* is used for singular objects, *-niit-* for plural objects). These suffixes are a transparent combination of the locative noun endings plus the verb stem *it-* ‘to be, exist’, and the same idea may be expressed as one word or two, as shown in example (8) (data from Kaplan [1981c:105] and MacLean [1986a:118]). In fact, the degree to which *-miit-* and *-niit-* are a single unit is debatable; Lawrence Kaplan (personal communication, 21 March 2011) characterizes it as a “superficial ‘run-on’” and notes that it is not possible in some other dialects.¹⁰

Ex. (8) a.	<i>iglumiittuq</i> <i>iglu-miit-tuq</i> house-be.located.in-IND.PRS.3SG ‘he/she/it is in the house’	b.	<i>iglumi</i> <i>ittuq</i> <i>iglu-mi</i> <i>it-tuq</i> house-LOC.SG be-IND.PRS.3SG ‘he/she/it is in the house’
------------	--	----	---

In both cases, it is clear that the /i/ from *it-* is /i/ because the following consonants are not palatalized (which would give **itchuq*). It appears, then, that /i/ may be realized as [i] when it follows another vowel, but not when it precedes another vowel (except in the rare cases discussed above).

2.1.2.3.2 ALTERNATION DUE TO GEMINATION OF A PRECEDING CONSONANT In Iñupiaq, gemination of underlying short consonants is morphologically conditioned (Kaplan 1981c:221–222); more specifically, certain allomorphs of certain morphemes, when attached to stems ending in $V_1C_1V_2(C_2)$, cause C_1 to geminate. Suffixes which trigger gemination and begin with a velar or uvular consonant cause an underlying /i/ in the V_2 position to be realized as [a] (Kaplan 1981c:121–124, 128–130):

¹⁰Woodbury (2002:86–87) discusses an analogous construction in Cup’ik, which, unfortunately, involves different phonological processes and so sheds no light on the question under consideration here.

- Ex. (9) a. *kammak*
kamik-k
 boot-ABS.DU
 ‘pair of boots’ (Kaplan 1981c:122)
- b. *siññaqturuq*
siñik-qtu-ruq
 sleep-excessively-IND.PRS.3SG
 ‘he/she sleeps a lot’ (MacLean n.d.a)
- c. *avvaq*
avik-q
 divide.in.half-result.of
 ‘a half’ (MacLean n.d.a)

Geminating suffixes beginning with phonemes other than a back consonant do not exhibit this alternation, although they may exhibit other alternations. Imperative suffixes *-iñ* 2SG and *-uŋ* 2SG>3SGO cause stem-final /i/ to be realized as [u] (Kaplan 1981c:125),¹¹ while postbase *-vik* ‘place or time for ____’ causes stem-final /i/ to be realized as [i] (Kaplan 1981c:131) (data from Kaplan [1981c:125, 131]):

- Ex. (10) a. *tiŋŋuñ*
tiŋi-iñ
 take.flight-IMP.2SG
 ‘take flight!’
- b. *attuŋ*
ati-uŋ
 put.on-IMP.2SG>3SGO
 ‘put it on!’
- Ex. (11) a. *suppivik*
supi-vik
 blow-place/time
 ‘break-up time’
- b. *tiŋŋivik*
tiŋi-vik
 take.flight-place/time
 ‘September (lit., time of taking flight)’

While the /i/-[u] alternation is limited to a handful of suffixes and not conditioned by any obvious phonological environment (Kaplan 1981c:132), the /i/-[a] alternation is clearly conditioned by the initial back consonant of the suffix triggering gemination (Kaplan 1981c:129). Kaplan suggests that the /i/-[a] alternation due to a following vowel is also conditioned by a back consonant, which would be the underlying first phoneme of the suffix following underlying /i/, and which would only appear in the surface form when preceded by two vowels and followed by a vowel (Kaplan 1981c:130). The status of this consonant will be considered in Section 2.1.2.4 on pages 14–16. For the present discussion, note that vowel clusters cannot contain [i], so [i] and this hypothetical underlying back consonant cannot co-occur in a surface form. If a back consonant is necessary to cause /i/ to alternate with [a], it is not sufficient; as shown in examples (12)–(14), suffixes beginning in a velar consonant preserve the underlying vowel quality of a preceding /i/ by default (data from MacLean [n.d.a]):

- Ex. (12) a. *īlisagigaluaqtaa*
īlisagi-kaluaq-taa
 recognize-without.desired.result-IND.PST.3SG>3SGO
 ‘he/she recognized him/her, but ...’

¹¹Kaplan (1981c:126) notes the existence of innovative versions of these morphemes which cause preceding /i/ to surface as [i].

- b. *isiǰaluaqtuaq*
isiq-kaluaq-tuaq
 enter-without.desired.result-IND.PST.3SG
 'he/she entered, but ...'
- Ex. (13) a. *ikayuqtikaaq*
ikayuqtī-kaa
 helper-usual
 'the one who usually helps'
- b. *inikaaq*
inī-kaa
 room/place-usual
 'usual place'
- Ex. (14) a. *iriqpaŋik*
iri-qpak-ŋik
 eye-big-ABS.DU.3SGP
 'his big eyes'
- b. *siñikpaksimaruaq*
siñik-qpak-sima-ruaq
 sleep-too.much-it.is.known-IND.PST.3SG
 'he/she overslept'

Thus, while back consonants are in part responsible for conditioning the /i/-[a] alternation that co-occurs with gemination, they do not condition this alternation by themselves (that is, in the absence of gemination). As for the /i/-[a] alternation before another vowel, the argument that a velar consonant is synchronically responsible for conditioning this alternation seems a stretch at best, given the following three observations: first, that, as demonstrated, velars generally do not have this effect in the synchronic phonology of Iñupiaq; second, that the velar in question rarely appears on the surface in any case, and never next to an [a] which is underlyingly /i/; third, that there is another, more obvious environment conditioning this alternation: a vowel following underlying /i/. I will therefore assume that, synchronically, the two environments conditioning the /i/-[a] alternation represent separate rules (probably stemming from a single historic origin).

2.1.2.4 Prevention of Three-Vowel Clusters

Iñupiaq does not allow clusters of more than two vowels to occur in the surface form of a word (MacLean 1986a:28; Kaplan 1981a:61). This restriction comes into play when a suffix which would begin with a vowel in the environments -C__ and -CV__ is attached to a stem ending in -VV (see examples (15) and (16)) or, if the suffix deletes stem-final consonants, -VVC (see example (17)).

- Ex. (15) *sumiuguvich*
su-miu-u-vît
 what-inhabitant.of-COP-INT.2SG
 ‘What place are you from?’
- Ex. (16) *nauganigmata*
nau-anik-mmata
 grow-already-CONSEQ.3PL
 ‘when they [i.e., fruit] had already
 grown’
- Ex. (17) *suagaqsigaa*
suak-aqsi-gaa
 scold-about.to-IND.PRS.3SG>3SGO
 ‘He/she is about to scold him/her’

In each of the examples above, the letter ⟨g⟩ between the two morphemes prevents a cluster of three vowels. In the case of the suffix *-a*, which marks possession in the absolutive case of a singular possessum by a third person singular possessor, the intervening consonant is not ⟨g⟩ but ⟨ŋ⟩ (Kaplan 1981c:190–192), as shown in (18). In (19), there are no potential three-vowel clusters, and ⟨ŋ⟩ is absent.¹² Data for both examples are from Kaplan (1981c:190, 192).

- Ex. (18) a. *kuuŋa*
kuuk-a
 river-ABS.SG.3SGP
 ‘his/her/its river’
- b. *puuŋa*
puuq-a
 sack-ABS.SG.3SGP
 ‘his/her/its sack’
- Ex. (19) a. *inaa*
ini-a
 place/room-ABS.SG.3SGP
 ‘his/her/its place/room’
- b. *savia*
savik-a
 knife-ABS.SG.3SGP
 ‘his/her knife’

The [ɣ] (⟨g⟩) and [ŋ] (in the case of the possessive suffix *-a*) which prevent three-vowel clusters may be analyzed as epenthetic or underlying. If epenthetic, they would be inserted whenever suffixation of a vowel-initial morpheme would create a three-vowel cluster; if underlying, they would be deleted except when deletion would create a three-vowel cluster (Kaplan 1981c:188–193). Kaplan prefers the underlying analysis, while MacLean adopts the epenthetic analysis (see MacLean 1986a:53). Kaplan’s reasoning is as follows:

- if one views the phenomenon as epenthesis, one needs two epenthesis rules (one for the possessive suffix and another for all other vowel-initial suffixes);
- the deletion analysis supports the hypothesis that the /i/-[a] alternation is conditioned by a velar (see Section 2.1.2.3, particularly pages 13–14);
- the word *iñuggun* ‘life’, nominal form of *iñuu-* ‘to be alive’, derived from *iñuk* ‘person’ and *-u-* COP, suggests the presence of a velar consonant at the beginning of the copular suffix, since the final consonant of *iñuk* would have been deleted, probably irretrievably, by the suffixation of *-u-*;

¹²Note, however, that a more innovative form of the ABS.SG.3SGP suffix is *-ŋa*; thus the forms *iniŋa* and *saviŋa* also exist (Kaplan 1981c:192). However, the forms **kuua* and **puua*, hypothetical homologues to *inaa* and *savia*, do not exist; the appropriate conservative forms of these words are *kuuŋa* and *puuŋa*.

- in today’s North Slope Iñupiaq, the ABS.SG.3SGP possessive suffix *-a* has been reanalyzed as *-ŋa* (the [ŋ] appearing in all surface forms), suggesting that the initial consonant was part of the conservative suffix (where it would have been deleted except to prevent three-vowel clusters).

From a purely synchronic perspective, however, the epenthetic analysis is attractive, despite its explanatory drawbacks. It assumes that suffixes which in most environments begin with a vowel also begin with a vowel underlyingly. Positing an underlying initial velar would require a distinction between “real” suffix-initial velars (as with the postbase *-gi-* ‘have as one’s ____’) and “deletable” ones (as with the copula *-(g)u-*); this is unnecessary under the epenthetic analysis. With the exception of the possessive ending *-a/-ŋa*, an epenthesis rule provides a simple yet adequate account of how Iñupiaq avoids three-vowel clusters. The possessive ending and other issues raised by Kaplan (1981c) can be accommodated in an epenthetic model as follows:

- the form *-ŋa* of the ABS.SG.3SGP possessive suffix can be considered an allomorph conditioned by the environment VV(C)___ (see Section 2.1.2.8 on pages 23–25). Note that the [ŋ] in question must be considered exceptional in both the epenthetic and underlying analyses; neither analysis explains why in this case alone the consonant would be nasal.
- the presence of velar consonants in *iñuggun* is evidence of a historic (lexicalized), rather than synchronic, phenomenon, and need not be accounted for here; like all other words whose derivation is not completely transparent, this word can simply be treated as a lexeme without regard to its etymology.
- the possibility of an underlying velar consonant conditioning the /i/-[a] alternation before another vowel was discussed in Section 2.1.2.3 on pages 13–14. While this alternation may have arisen due to a historic velar consonant, it is much easier to assume that synchronically the alternation is triggered by a preceding vowel than by a preceding disappearing velar.

For present purposes, another advantage of the epenthetic analysis is that it makes it easier to convert MacLean’s materials into a computational morphology, since she treats the phenomenon as epenthesis. For these reasons, and because my goal is more descriptive than explanatory, I adopt the epenthetic analysis in the present work.

2.1.2.5 Gemination

As mentioned in Section 2.1.2.3, gemination of underlying short consonants is morphologically conditioned in Iñupiaq (Kaplan 1981c:221–222). When a suffix which triggers gemination is attached to a stem ending in $V_1C_1V_2(C_2)$, underlying C_1 is realized on the surface as its long counterpart. In most cases, this counterpart is simply the phonetic geminate of C_1 , but /s/ geminates to [tʃ:].

MacLean’s treatment of Iñupiaq morphophonology (e.g., 1986a; 1986b; n.d.b; n.d.a) ascribes the ability to trigger gemination to allomorphs rather than to morphemes. This provides a straightforward mechanism to deal with the fact that “gemination usually depends in some way upon the final segment of the stem and does not occur categorically before a certain morpheme...” (Kaplan 1981c:225). For example, the suffix *-vik* ‘place, time’ causes gemination only in stems ending in a vowel (example (20); data from MacLean [n.d.a]), while singular relative case marker *-(u)m* causes gemination only in stems ending in /q̃/ (example

(21); data from (MacLean 1986a:148)). MacLean analyzes *-vik* as having two allomorphs, identical in form, one of which follows vowels and triggers gemination, the other of which follows consonants and does not trigger gemination. Similarly, the singular relative case marker, *-(u)m*, is analyzed with three allomorphs: *-m* which does not allow gemination occurs after vowels, *-m* which does allow gemination occurs after /*q̃*/, and *-um* (which does not allow gemination) occurs after “strong” consonants. I adopt this approach in the present work. An alternative to this approach would be to specify allomorphy and gemination patterns separately. Such an approach would be more elegant in the case of morphemes such as the relative singular ending, which could be analyzed as having two allomorphs, one of which exhibits two distinct gemination patterns, rather than three allomorphs, two of which have identical forms but disparate gemination patterns. However, specifying separate conditioning environments for allomorphy and gemination would require additional notational complexity, which makes the approach less attractive for purposes of computational morphology. Also, because MacLean’s work is the main source of lexical data for this thesis, adopting her treatment of gemination simplifies the lexicon creation process.

Ex. (20) a.	<i>niġġivik</i> <i>niġi-vik</i> eat-place ‘table’	Ex. (21) a.	<i>ammim</i> <i>amiq̃-m</i> animal.hide-REL.SG ‘(of) the animal hide’
b.	<i>annivik</i> <i>ani-vik</i> birth-time ‘birthday’	b.	<i>qayyam</i> <i>qayaq̃-m</i> kayak-REL.SG ‘(of) the kayak’
c.	<i>naġirvik</i> <i>naġit-vik</i> sick-place ‘hospital’	c.	<i>inim</i> <i>ini-m</i> place-REL.SG ‘(of) the place’
d.	<i>savagvik</i> <i>savak-vik</i> work-place ‘office, place of employment’	d.	<i>tupqum</i> <i>tupiq̃-um</i> tent-REL.SG ‘(of) the tent’

Many allomorphs allow but do not require gemination. The relative singular case ending is one such suffix. Thus, the forms *ammim* ‘(of) the animal hide’ and *qayyam* ‘(of) the kayak’ given in example (21) exist alongside innovative non-geminated forms *amim* and *qayam* (MacLean 1986a:148; Lawrence Kaplan, personal communication, March 21, 2011).

In certain stems, the consonant resulting from gemination is unpredictable from the form of the stem. These unpredictable geminates fall into two categories: stops which correspond to continuants in the ungeminated stem (see example (22), data from MacLean [1986a:93]), and intervocalic voiced fricatives that correspond to zero in the ungeminated stem (see example (23), data from MacLean [1986b:25, 107]). In all cases, phonological change is responsible for these unusual geminates. [j] which geminates to [tʃ:] is likely due to a historic alternation between [j] and [tʃ] (Kaplan 1981c:179). All others are due to a diachronic process of consonant gradation whereby (among other changes) ungeminated stops became

voiced fricatives and ungeminated voiced fricatives were deleted, while in their geminated forms the consonants retained their historic form (Kaplan 1982).

- | | |
|---|--|
| <p>Ex. (22) a. <i>q̄l̄aluk̄k̄ak</i>
 <i>q̄l̄alugaq-k</i>
 beluga-ABS.DU
 '(two) beluga whales'</p> <p>b. <i>p̄ap̄iq̄q̄uk</i>
 <i>p̄ap̄iq̄uq-k</i>
 fishtail-ABS.DU
 '(two) fishtails'</p> <p>c. <i>q̄at̄chut</i>
 <i>q̄ayuuq-t</i>
 broth-ABS.PL
 'pots of broth'</p> | <p>Ex. (23) a. <i>paḡḡiiñ</i>
 <i>paḡḡi-iñ</i>
 stay-IMP.2SG
 'stay at home!'</p> <p>b. <i>qaḡḡiiñ</i>
 <i>qaḡḡi-iñ</i>
 come-IMP.2SG
 'come here!'</p> <p>c. <i>tagiḡḡirirut</i>
 <i>tagiḡḡuq-i-rut</i>
 salt-make-IND.PRS.3PL
 'they are producing salt'</p> |
|---|--|

2.1.2.6 Suffix Attachment Patterns

Many of the morphophonological changes that occur at morpheme boundaries are conditioned by the suffix being attached. Although the pattern of attachment depends to some degree on the initial phoneme of the suffix, it is not entirely predictable from phonological form, as demonstrated by postbases *-saḡataq-* 'for a long time', *-sugruk-* 'a lot', and *-siññaq-* 'only'. *-saḡataq-* attaches directly to the stem without deleting any segments (see example (24) below); *-sugruk-* deletes any stem-final consonant (example (25)); and *-siññaq-* deletes stem-final /t/ but not /k/ or /q/ (example (26)) (data for examples (24)–(26) from MacLean [n.d.a]).

- | | |
|--|--|
| <p>Ex. (24) a. <i>aq̄pat̄chaḡataḡuuruq</i>
 <i>aq̄pat-saḡataq-uu-ruq</i>
 run-for.a.long.time-usually-IND.PRS.3SG
 'he/she usually runs for a long time'</p> <p>b. <i>savaks̄aḡataḡuuruq</i>
 <i>savak-saḡataq-uu-ruq</i>
 work-for.a.long.time-usually-IND.PRS.3SG
 'he/she usually works for a long time'</p> <p>c. <i>niḡisaḡataqtuq</i>
 <i>niḡi-saḡataq-tuq</i>
 eat-for.a.long.time-IND.PRS.3SG
 'he/she eats for a long time'</p> | <p>Ex. (25) a. <i>aq̄pasugruktuḡa</i>
 <i>aq̄pat-sugruk-tuḡa</i>
 run-a.lot-IND.PST.1SG
 'I ran a lot'</p> <p>b. <i>uqaḡsugrugnak</i>
 <i>uqaq-sugruk-nak</i>
 talk-a.lot-NEGCONTEMP.2SG
 'don't talk too much'</p> <p>c. <i>niḡisugruktuḡagut</i>
 <i>niḡiḡ-sugruk-tuḡagut</i>
 eat-a.lot-IND.PST.3PL
 'we ate a lot'</p> |
|--|--|

- Ex. (26) a. *aqvısiññaqtuaq*
aqvıt-siññaq-tuaq
 sit-only-IND.PST.3SG
 ‘he/she just sat down (he did nothing else)’
- b. *uqaqsiññaqtuat*
uqaq-siññaq-tuat
 talk-only-IND.PST.3PL
 ‘they just talked (they did nothing else)’
- c. *napişiññaqtugıch*
napi-siññaq-tugıch
 cut.in.half-only-CONTEMP1.REAL.3PLO
 ‘they having been cut (just) in half’

MacLean, in her dictionaries and pedagogical grammar books, identifies six¹³ main attachment patterns (1981:xi; 1986a:261; 1986b:102, 105). Rather than name the patterns, she assigns them symbols. The patterns are listed in Table 2.4; examples follow. Discussion continues on page 22.

TABLE 2.4: MacLean’s suffix attachment patterns

Symbol	Explanation
–	Delete any stem-final consonant (see example (25) on the preceding page)
+	Attach directly to stem (see example (24) on the preceding page); stem-final stops become voiced fricatives if the suffix begins with a voiced phoneme (see example (27) on the following page); if the stem ends in a consonant and the suffix begins with a consonant cluster, the first consonant of the suffix is deleted (see example (28) on the next page)
:	<i>For stems ending in VC₁iC₂</i> : delete /i/; assimilate C ₂ to agree with C ₁ in terms of voicing, continuancy, and optionally nasality (see example (29.a) on the following page) <i>For stems ending in VC₁C₂iC₃</i> : C ₃ becomes a voiced fricative (see example (29.b) on the next page); alternatively, some speakers delete C ₃ and treat /i/ as /i/ (MacLean 1986a:52) <i>For stems ending in VC where V is a vowel other than /i/</i> : delete C (see example (29.c) on page 21) <i>For stems ending in V</i> : attach the stem directly (/i/ becomes [a] as described in Section 2.1.2.3 on pages 11-14; potential three-vowel clusters are broken up by a [ɣ] ((g)) as described in Section 2.1.2.4 on pages 14-16) (see example (29.d) on page 21)
÷	delete stem-final /q̄/ (see example (30) on page 21)
±	delete any stem-final back consonant (see example (31) on page 21)
≠	delete any stem-final /t/ (see examples (26) on the preceding page and (32) on page 22)

¹³Two additional patterns, denoted with the symbols ‘=’ and ‘~’ are omitted from the present discussion. These are minor patterns which occur only with a handful of suffixes and only in restricted phonological environments, and allomorphs exhibiting these patterns often have slightly different semantics from other allomorphs. These facts suggest that these patterns are not synchronically productive.

The ‘-’ pattern was illustrated in example (25) on page 18. Suffixes exhibiting this pattern cause the deletion of the final consonant of the stem to which they attach. As described in Section 2.1.2.4 on pages 14-16, potential three-vowel clusters created by this deletion are broken up by a [χ] ((g)) inserted before the initial vowel of the suffix.

Examples (27) and (28) illustrate the ‘+’ pattern (see also (24) on page 18). Postbase *-ruiññaq-* ‘finally’ causes regressive assimilation of stem-final consonants; postbase *-qpak(-)* ‘a big ____; to ____ a lot, excessively’ begins with two consonants, the first of which is deleted when attached to a consonant-final stem. Data for both examples come from MacLean (n.d.a).

<p>Ex. (27) a. <i>aaǰluǰruiññaqtuq</i> <i>aaǰluq-ruiññaq-tuq</i> raise.head-finally-IND.PRS.3SG ‘it finally raised its head’</p> <p>b. <i>avirruññaqtuak</i> <i>avit-ruiññaq-tuak</i> get.a.divorce-finally-IND.PRS.3DU ‘they finally got a divorce’</p> <p>c. <i>katagruiññaqtuq</i> <i>katak-ruiññaq-tuq</i> fall-finally-IND.PRS.3SG ‘he/she finally fell’</p>	<p>Ex. (28) a. <i>nigíqpagnak</i> <i>nigi-qpak-nak</i> eat.a.lot-NEGCONTEMP.2SG ‘don’t eat too much’</p> <p>b. <i>umiaqpak</i> <i>umiaq-qpak</i> boat-big ‘ship (lit., big boat)’</p> <p>c. <i>savikpaŋa</i> <i>savik-qpak-ŋa</i> knife-big-ABS.SG.3SGP ‘his sword (lit., big knife)’</p>
--	--

Example (29) exemplifies the complex ‘:’ pattern, which affects stems differently depending on their final phonemes. Stems ending in VC₁iC₂ undergo the most drastic change; /i/ is deleted, creating a consonant cluster in which the second consonant undergoes assimilation for voicing, continuancy, and nasality (29.a). In stems ending in VC₁C₂iC₃, deletion of /i/ would create a three-consonant cluster, which is not allowed in Iñupiaq; instead, the final consonant becomes a voiced fricative (29.b). With stems ending in VC where V is a vowel other than /i/, the ‘:’ pattern is identical to the ‘-’ pattern: the stem final consonant is deleted (29.c). With vowel-final stems, the ‘:’ pattern is identical to the ‘+’ pattern; stem-final /i/ becomes [a] (see Section 2.1.2.3 on pages 11-14) and potential three-vowel clusters are broken up by a [χ] ((g)) (see Section 2.1.2.4 on pages 14-16) (29.d). It’s worth noting that all suffixes exhibiting the ‘:’ pattern begin with a vowel. The postbase used in the example below is *-ît-* ‘to lack ____; to not be ____’; data are from MacLean (1986a:51-53).

<p>Ex. (29) a. i. <i>kamŋitchuq</i> <i>kamiĕ-ît-tuq</i> boot-lack-IND.PRS.3SG ‘he has no boots’ / ‘he is not wearing boots’ / ‘there are no boots’</p>	<p>ii. <i>tupqitchuŋa</i> <i>tupîĕ-ît-tuŋa</i> tent-lack-IND.PRS.1SG ‘I have no tent(s)’</p>
---	---

- | | |
|--|---|
| <p>b. <i>iġñiġitchuq</i>
 <i>iġniġ-ît-tuq</i>
 son-lack-IND.PRS.3SG
 ‘he has no son(s)’</p> <p>c. <i>miqliqtuitchuq</i>
 <i>miqliqtuq-ît-tuq</i>
 child-lack-IND.PRS.3SG
 ‘he/she has no child(ren)’ / ‘there are no children’</p> | <p>d. i. <i>atigaitchuġa</i>
 <i>atigi-ît-tuġa</i>
 parka-lack-IND.PRS.1SG
 ‘I have no parka(s)’ / ‘I am not wearing a parka’</p> <p>ii. <i>quagitchuġa</i>
 <i>quaq-ît-tuġa</i>
 frozen.meat-lack-IND.PRS.1SG
 ‘I have no frozen meat’</p> |
|--|---|

The ‘÷’ pattern is illustrated in example (30) using the terminalis singular ending *-mun* ‘to ____ (which is singular)’. This pattern is restricted to a handful of nasal-initial nominal inflectional endings and the derivational suffixes that are derived from these endings. This pattern involves the deletion of stem-final /ġ/; any other underlying stem-final phoneme is retained. The data presented below are from MacLean (1986a:115–117).

- | | |
|--|--|
| <p>Ex. (30) a. <i>iglumun</i>
 <i>iglu-mun</i>
 house-TRM.SG
 ‘to the house’</p> <p>b. <i>tupiġmun</i>
 <i>tupiġ-mun</i>
 tent-TRM.SG
 ‘to the tent’</p> | <p>c. <i>qimmimun</i>
 <i>qimmîġ-mun</i>
 dog-TRM.SG
 ‘to the dog’</p> |
|--|--|

The ‘±’ pattern, shown in example (31), deletes stem-final back consonants only. Suffix *-kasak-* ‘regularly, routinely’ exhibits this pattern (data from MacLean [n.d.a]).

- | | |
|--|--|
| <p>Ex. (31) a. <i>niġikasaktuani</i> <i>uvaptinni</i>
 <i>niġi-kasak-tuani</i> <i>uvaptit-ni</i>
 eat-regularly-IND.PST.1PL we/us-LOC
 ‘we’ve been eating regularly at our place’</p> <p>b. <i>aqpatkasaktuagni</i>
 <i>aqpat-kasak-tuagni</i>
 run-regularly-IND.PST.1DU
 ‘the two of us have been running regularly’</p> | <p>c. <i>ikayukasaktaġa</i>
 <i>ikayuq-kasak-taġa</i>
 help-regularly-IND.PST.1SG>3SGO
 ‘I’ve been helping him/her regularly’</p> |
|--|--|

The ‘≠’ pattern deletes only /t/ (the only underlying stem-final coronal consonant allowed in Iñupiaq phonotactics). This pattern is exemplified in (32) below using the suffix *-tilaaq(-)* which expresses (among other things) measurement of a quality or performance of an action to the greatest extent possible (data from MacLean [n.d.a]). See also example (26) on page 18.

- Ex. (32) a. *akisutīlāaq*
akisu-tilāaq-∅
 be.valuable-measure-ABS.SG
 ‘price’
- b. *pisuqtilāaḡlusik*
pisuq-tilāaq-lusik
 be.good.at-to.one’s.ability-CONTEMP1.UNREAL.2DU
 ‘you (two) doing your best’
- c. *uqumaisilāaq*
uqumait-tilāaq-∅
 be.heavy-measure-ABS.SG
 ‘weight’

As with the conditioning of gemination (see 2.1.2.5 on page 16), MacLean (e.g., 1986a; 1986b; n.d.b; n.d.a) treats suffix attachment patterns as a property of individual allomorphs rather than of morphemes. This approach is clearly necessary for morphemes which exhibit widely divergent allomorphy, such as the suppletive pair $\mp tit-/+pkaq-$ which express causation: *-tit-* occurs following consonants, while *-pkaq-* follows vowels. But this method is also practical for describing the situation of suffixes which exhibit more than one attachment pattern. For example, diminutive morpheme *-uraq* usually exhibits the ‘:’ pattern, but attaches to stems ending in VVC according to the ‘=’ pattern; treating this morpheme as having two allomorphs, *:uraq* and *=uraq*, provides a straightforward way to describe this behavior.

2.1.2.7 /t̥i/-[n] Alternation

As a general rule, the absolutive singular form of an Iñupiaq noun is taken as that noun’s stem. But this appears not to be the case for absolutive singular nouns ending in [n] or its palatal allophone, [ɲ] ((ñ)). With these nouns, the final nasal of the absolutive singular form corresponds to [t̥i] ([s̥i] for (ñ)) before most suffixes other than enclitics. This is illustrated in examples (33) and (34) below.

- Ex. (33) a. *aḡun*
aḡut̥i-∅
 man-ABS.SG
 ‘man’
- b. *aḡut̥ik*
aḡut̥i-k
 man-ABS.DU
 ‘two men’ (MacLean 1986a:77)
- c. *aḡut̥im*
aḡut̥i-m
 man-REL.SG
 ‘(of) the man’ (MacLean 1986a:148)
- d. *aḡut̥auruq*
aḡut̥i-u-ruq
 man-COP-IND.PRS.3SG
 ‘it is a man’ (MacLean n.d.a)
- e. *aḡut̥igik-*
aḡut̥i-g̥ik-
 man-beautiful
 ‘to be handsome’ (MacLean n.d.a)
- Ex. (34) a. *ak̥iñ*
ak̥isi-∅
 pillow-ABS.SG
 ‘pillow’
- b. *ak̥isik*
ak̥isi-k
 pillow-ABS.DU
 ‘two pillows’ (MacLean 1986a:77)
- c. *ak̥isim*
ak̥isi-m
 pillow-REL.SG
 ‘(of) the pillow’ (MacLean 1986a:148)

There are, however, a handful of exceptional suffixes which allow [n] or [ɲ] in the surface form instead of [ti] or [si]. These are the singular inflectional endings of the following cases: modalis (-mik), locative (-mi), ablative (-miñ), and terminalis (-mun) (MacLean 1986a:97, 115). Thus, forms like *aḡunmun* ‘to the man’ and *akiñmi* ‘on the pillow’ are allowed (as are *aḡutimun* and *akisimi*). The fact that noun stems exhibiting this alternation tend to appear with [ti] by default and with [n] only before certain suffixes points to [ti] as the underlying form (a conclusion implicit in Kaplan [1981c]).

The /ti/–[n] alternation is clearly morphologically conditioned (Kaplan 1981c:147), but while all absolutive singular nouns ending in ⟨n⟩ or ⟨ñ⟩ exhibit the alternation described here (Kaplan 1981c:149), the converse is not true. That is, there are noun stems ending in /ti/ or /si/ which do not alternate with ⟨n⟩ or ⟨ñ⟩, even in the presence of suffixes which trigger this alternation. Examples of stems not exhibiting the /ti/–[n] alternation include *aglakti* ‘secretary,’ *aḡaiyyuliqsī* ‘preacher,’ *aḡpati* ‘runner,’ *pisiksī* ‘bow (hunting implement),’ *sisi* ‘burrow, den’ (Kaplan 1981c:147; MacLean 1981). Stems ending in VCCī, such as *aglakti* and *pisiksī*, could not undergo this alternation due to phonotactic constraints (a stem cannot end in two consonants; see page 26), but these constraints cannot explain why *aḡpati* and *sisi* do not exhibit the alternation (the Kobuk equivalent of *sisi* actually does undergo this alternation, becoming *siñ* [Lawrence Kaplan, personal communication, 5 April 2011]). Stems which undergo the alternation must be specifically marked as such; specifically, the stems subject to this alternation all end with the instrumental nominalizing morpheme *-uti*, etymologically if not synchronically (Kaplan 1981c:147–148). Thus, the alternation is conditioned by both morphological and lexical criteria; underlying /ti/ is only realized as [n] when the stem ends in *-uti* and is followed by a suffix which triggers this alternation (Kaplan 1981c:148).

Although /ti/ and /si/ more accurately reflect the underlying forms of stems which exhibit the /ti/–[n] alternation, glosses in this document (except in examples (33) and (34) in this section) use the form with [n] instead. This is done for two reasons: first, because readers aware of this alternation will recognize that a stem-final ⟨n⟩ is underlyingly /ti/, whereas they cannot know *a priori* whether a given /ti/ alternates with [n]; and second, because the form ending with a nasal, as the absolutive singular form, is conventionally used as the citation form in Iñupiaq dictionaries and grammar texts. Using this form may make this work more accessible to those familiar with existing Iñupiaq language resources.

2.1.2.8 Allomorphy

I have already discussed allomorphy in the contexts of gemination (see pages 16–17) and suffix affixation patterns (see preceding page). These types of allomorphy stretch the traditional definition of “allomorph” in that the allomorphs in question often differ from each other not in phonetic form, but rather in the effects they have on the stems to which they attach.

That is not to say that Iñupiaq lacks allomorphy in the more conventional sense of the term. As Kaplan (1981c:232) writes, “a great many Inupiaq suffixes exhibit allomorphy for which no one proposes a synchronic phonological account.” Common allomorphic variations include voiceless vs. voiced initial consonant and presence or absence of an initial consonant; less common variations include presence or absence of an initial vowel, syllable, or consonant cluster. There is also at least one productive case of suppletive allomorphy: the causative suffixes *-tit-* and *-pkaq-*.

For the most part, productive allomorphy in Iñupiaq is phonetically conditioned, usually by the final seg-

ment or syllable of the stem, but occasionally by longer strings of phonemes. For the pairs *-pallîq-/vallîq-* ‘appears to be, is probably, sounds like one is ____ing’ in example (35) and *-sima-/ma-* ‘it is now know that one has ____ed, is ____ing, or is ____ed’ in example (36), the first allomorph is used for consonant-final stems, and the second for vowel-final stems. With the pair *-suk-/uk-* ‘to want to ____’ in example (37), the first allomorph is used for stems ending in a vowel or {t}, the second for stems ending in a back consonant. Data for all three examples is from MacLean (n.d.a).

- | | |
|---|--|
| <p>Ex. (35) a. <i>siñik^hpallîqsut</i>
 <i>siñik-pallîq-tut</i>
 sleep-probably-IND.PRS.3PL
 ‘they are probably sleeping’</p> <p>b. <i>kasimavallîqsut</i>
 <i>kasima-vallîq-tut</i>
 hold.a.meeting-probably-IND.PRS.3PL
 ‘they are probably having a meeting’</p> | <p>Ex. (37) a. <i>îlausuk^hpit</i>
 <i>îlau-suk-pît</i>
 be.included-want-INT.2SG
 ‘do you want to be included?’</p> <p>b. <i>makitchuk^hpisi</i>
 <i>makit-suk-pisi</i>
 stand.up-want-INT.2PL
 ‘would you all like to stand up?’</p> <p>c. <i>tautuguk^hkiga</i>
 <i>tautuk-suk-kiga</i>
 see-want-IND.PRS.1SG>3SGO
 ‘I would like to see it’</p> |
| <p>Ex. (36) a. <i>paqisimaraa</i>
 <i>paqît-sima-raa</i>
 find-it.is.known-IND.PST.3SG>3SGO
 ‘he/she did find it’</p> <p>b. <i>naatchimaruat</i>
 <i>naatchî-ma-ruat</i>
 finish-it.is.known-IND.PST.3PL
 ‘they did finish’</p> | |

An example of more complex allophonic conditioning is the second person imperative ending *-îñ*, which has four allomorphs: *-îñ* which triggers gemination attaches only to stems of the form (C)VCV (see example (38)); *-îñ* which does not trigger gemination attaches to stems ending in a back consonant and stems longer than two syllables which end in a vowel (see example (39)); *-tîñ* attaches to stems ending in /t/ (see example (40)); and *-ggiñ* attaches to stems which end in a two-vowel cluster (see example (41)) (MacLean 1986b:27–28, including examples).

- | | |
|---|--|
| <p>Ex. (38) a. <i>niğgiñ</i>
 <i>niğî-îñ</i>
 eat-IMP.2SG
 ‘eat!’</p> <p>b. <i>aggiñ</i>
 <i>a^gî-îñ</i>
 go.home-IMP.2SG
 ‘go home!’</p> | <p>Ex. (39) a. <i>savagiñ</i>
 <i>savak-îñ</i>
 work-IMP.2SG
 ‘work!’</p> <p>b. <i>iglañaiñ</i>
 <i>iglaña-îñ</i>
 smile-IMP.2SG
 ‘smile!’</p> |
|---|--|

- Ex. (40) a. *makittin̄*
makit-tin̄
 stand.up-IMP.2SG
 ‘stand up!’
- b. *aqattin̄*
aqat-tin̄
 run-IMP.2SG
 ‘run!’

- Ex. (41) a. *iglau^gguin̄*
iglau-gguin̄
 be.moving-IMP.2SG
 ‘keep moving!’
- b. *naku^gguin̄*
nakuu-gguin̄
 be.good-IMP.2SG
 ‘be good!’

2.1.3 Phonotactics

2.1.3.1 Consonant Clusters

Consonant clusters contain exactly two phonemes (Kaplan 1981c:32; MacLean 1986a:28). For purposes of this discussion, a long consonant may be considered a consonant cluster. [f], [ʂ], [x], [χ], [h], [ŋ], and [s] may not occur as long consonants (Kaplan 1981c:32) (underlying [s] geminates to [ʃ:]; the other consonants just listed do not undergo gemination).

The process of regressive assimilation is responsible for many of the surface constraints on consonant clusters (see section 2.1.2.1 on pages 7–8): adjacent consonants must share the same voicing and continuancy; a cluster-initial consonant other than a velar or uvular must match the nasality of the following consonant (in some dialects, the nasality constraint applies also to back consonants), and a cluster-initial alveolar, palatal, or retroflex consonant must agree in laterality with the following consonant (Kaplan 1981c:39–41).

A consonant cluster cannot contain both a velar and a uvular (MacLean 1986a:29), nor can it contain both an alveolar and a palatal (in consequence of the palatalization rule Kaplan [1981c:85]; see section 2.1.2.2 on pages 9–11). [s] is considered a palatal for the purposes of this constraint (see page 11).

2.1.3.2 Vowel Clusters

As discussed in Section 2.1.2.4 on pages 14–16, vowel clusters are limited to two vowels (MacLean 1986a:28; Kaplan 1981a:61). Except for the cases mentioned on page 12 in Section 2.1.2.3, /i/ cannot appear as such in a vowel cluster; all other combinations of vowels are allowed (Kaplan 1981c:31–32).

2.1.3.3 Morpheme and Word Structure

Both entire words and individual morphemes in Iñupiaq consist of alternating vowel groups and consonant groups (where a group is one or two phonemes of the same category (consonant or vowel) bounded on each side either by an edge or by a phoneme of the opposite category). A word minimally consists of a single vowel group.

Stems in North Slope Iñupiaq may begin with a vowel or vowel cluster¹⁴ or with any of the following

¹⁴All possible vowel clusters are well attested stem-initially except [iu], which to my knowledge is unattested, and [ia], for which MacLean (n.d.a) lists only the exclamation *iaqi* ‘don’t!’.

consonants: [p], [t], [k], [q], [s], [m], [n], and rarely, [y], [l], and [v] (Kaplan 1981c:32; MacLean 1986a:28). Interjections may begin with other consonants, for example, *hauk* (exclamation of fatigue), *ñiaq* 'don't!' (Kaplan 1981c:32-33; MacLean n.d.a). Stems may not begin with a consonant cluster (Kaplan 1981c:32; MacLean 1986a:28).

Noun stems and verb stems allow different stem-final phonemes. Verb stems may end in [q], [k], [t], [tʃ̥] (as a palatalized allophone of [t]), or a vowel; noun stems may end in [q], [k], or a vowel (Kaplan 1981c:34). Nouns in the absolutive singular may also end in [n] or [ɲ] ([ñ]); however, these consonants correspond to underlying /t̥/ and /s̥/ (see section 2.1.2.7 on pages 22-23). For both noun stems and verb stems, stem-final vowel clusters are permitted, but stem-final consonant clusters are not (Kaplan 1981c:32; MacLean 1986a:28).

A stem must contain at least one vowel; beyond that, in theory, stems may be arbitrarily long.

Suffixes, including enclitics, may begin with a vowel cluster, a single vowel with the exception of /i/ (Kaplan 1981c:105-119), a consonant cluster, or a single consonant. Suffixes and words in general may end with a vowel cluster, a vowel, or a single consonant. The same constraints that apply at the right edge of noun stems also apply at the right edge of suffixes which derive nouns: if they end in a consonant, that consonant must be [q] or [k]. Likewise, suffixes which derive verbs are subject to the same right-edge constraints that apply to verb stems (the only consonants allowed are [q], [k], [t], and [tʃ̥]). Words and inflectional suffixes may end with any of the following consonants: [t], [tʃ̥], [k], [q], [m], [n], [ɲ], and [ŋ] (Kaplan 1981c:33; MacLean 1986a:28).¹⁵

Except for enclitics, suffixes need not contain a vowel group, as long as they satisfy all of the conditions for initial and final phonemes; like stems, suffixes may theoretically be arbitrarily long, although the vast majority contain no more than two vowel groups.

2.1.3.4 Distribution of Specific Phones

As discussed on page 11, [tʃ̥] and [s] occur in complementary distribution: [s] may only occur word-initially, at the end of a consonant cluster, and between vowels as a short consonant, while [tʃ̥] may only occur word-finally, at the beginning of a consonant cluster, and between vowels as a long consonant (Kaplan 1981c:86).

[i] may not follow word-initial [n] or any [t] (word-initial or otherwise) (Kaplan 1981c:91-92). [i] may not follow [s] unless that [s] is underlyingly /t/; word-initial [s] is always /s/ underlyingly (Kaplan 1981c:91-93). Aside from the handful of exceptions given on page 12 in Section 2.1.2.3, /i/ cannot occur in a vowel cluster (Kaplan 1981c:118-121). As was mentioned in Section 2.1.3.2 on the preceding page, /i/ cannot be the initial phoneme of a suffix (Kaplan 1981c:105, 119).

In consequence of palatalization, [ɲ], [ʎ], [ʂ], and short [tʃ̥] can only occur in the environment $\hat{i}(C)_$ (see Section 2.1.2.2 on pages 9-11).

Any /q/ at the end of a noun stem is either weak (\check{q}) or strong (\hat{q}) (see page 6 in Section 2.1.1.1). Only / \hat{q} / is allowed in the environment $\check{i}_$. Only / \check{q} / is allowed in the environment $VCV_1(V)_$ (where V_1 is a vowel other than /i/). In other environments, the lexicon determines whether each instance of stem-final

¹⁵There are no noun inflections which end in [q], though many absolutive singular nouns do. There are no verb inflections which end in [m] aside from participial and gerundive endings.

/q/ is weak or strong (MacLean 1986a:76-77).

2.1.4 Morphology

The discussion in this section will draw on work pertaining to several languages within the Eskimo family and several dialects within the Inuit branch. This is possible because the various Eskimo languages have very similar morphologies (Woodbury 2002:80; Tersis and Therrien 2000:20). The shared properties of particular interest in this section are a common set of basic grammatical categories (e.g., noun, verb, pronoun, demonstrative, etc.) and a common word structure (that is, a stem followed by zero or more postbases, followed by inflection, followed by zero or more enclitics). These are explored in more detail below.

2.1.4.1 Grammatical Categories

Eskimo grammatical categories can be grouped according to whether or not they allow inflection (Mithun 2000:85; Woodbury 2002:81-84; Miyaoka 2000:226-227). The class of uninflected words, often lumped together as “particles,” may be further divided into interjections, conjunctions, adverbs, enclitics (which are phonologically (and orthographically) suffixed to the preceding word but which function syntactically as separate words [Woodbury 2002]) and perhaps other minor categories (Fortescue 2004:1390; Woodbury 2002:84; Miyaoka 2000:227). The class of inflected grammatical categories can be divided into nominal and verbal subcategories (or, in terminology-neutral analyses such as Lowe (1996, 2000) and Cornillac (2000), words denoting beings or substances and words denoting processes or states).

All nominals are inflected for some subset of the following: case, number, and possessor if applicable. The largest, most prototypical subset of nominals, which we might call “nouns proper” or just “nouns”, is inflected for all three properties. Personal pronouns are also considered nominal (e.g., Fortescue 2004:1390); these are inflected for case and number and reflect grammatical person. Fortescue (2004:1390) also treats demonstratives as nominal; Jacobson (1984:653) and MacLean (1986a:227) distinguish between demonstrative pronouns and demonstrative adverbs (without placing either set of demonstratives within or outside any particular category). Demonstrative pronouns are inflected for number and case, while demonstrative adverbs may only be inflected for case, and even then only for those cases expression location or direction (note also that zero-marked demonstratives are treated by MacLean and Jacobson as “interjectional,” in contrast to zero-marked nouns, which are analyzed as “absolute”). If one considers demonstrative adverbs nominals, they are the most atypical members of the set.

All members of the verbal category can be considered “verbs proper”; we can, however, recognize a distinction between words whose inflections reflect a direct object—transitive verbs—and words whose inflections do not—intransitive verbs. Verbs are inflected for mood, grammatical number and person of subject, and grammatical number and person of direct object if applicable.

Only nouns and verbs are open categories in Iñupiaq.

2.1.4.2 Categories of Word Formatives

Most Eskimo word formatives¹⁶ can be classed into one of three main categories, although different analyses may refer to these using slightly different terminology. I will use the terms base, postbase, and inflection, which are perhaps the most common terms in the literature on Eskimo morphology. Bases (sometimes also called roots or stems) are free forms; postbases are generally thought of as derivational suffixes, although some are grammatical in nature (Fortescue 2004:1393-1394); and inflectional endings are paradigmatic grammatical suffixes. Other formative categories include enclitics and “reduced” (cliticized) words, post-inflectional derivational suffixes, and the demonstrative prefix *ta(t)-*. These will be discussed later in this section.

Just as Iñupiaq words can be classified as nouns, verbs, and so on, bases, postbases, and inflectional endings may be classified into grammatical categories. The classification scheme developed here is based on ideas expressed in Lowe (2000), but unlike Lowe, I employ traditional part-of-speech terms from the Indo-European grammatical tradition. I note the objections of Cornillac (2000:178-179) regarding this practice: that such terms obscure the significant differences between Indo-European and Eskimo morphosyntax, particularly the fact that the nominal or verbal status of an Eskimo lexeme in no way predicts the nominal or verbal status of the word containing that base. While Cornillac’s point is well-taken, I find that terms such as “noun” and “verb” convey in a concise way important attributes of these formatives that could otherwise only be awkwardly expressed. Similar conventions are used by many Eskimologists (see for example Fortescue 1980, 2004; MacLean 1986a:42; Woodbury 2002:84).

One could devise a complex set of rules, including semantic and syntactic considerations, for classifying Eskimo formatives, but for present purposes I rely primarily on a few simple morphological criteria. I assume that each paradigmatic set of inflectional endings is trivially identified as pertaining to nouns, intransitive or transitive verbs, personal pronouns, or demonstratives. A base may then be classified according to the inflectional paradigms that apply to it. For example, the base *aqpat-* ‘to run (of a human)’ takes verbal inflectional endings such as *-tuq*, *-vluŋa*, and *-kuvît*, and thus can be classified as a verb base; the base *nanuq* ‘polar bear’ takes nominal inflectional endings such as *-t*, *-nik*, and *-tun*, and thus can be classified as a noun base. In addition to nouns and verbs, I recognize demonstrative bases, pronoun bases, and particles (members of this last set cannot be inflected). As Iñupiaq has distinct intransitive and transitive inflectional paradigms, verb bases may be further categorized as either intransitive, transitive, or ambitransitive (Mithun 2000:86-87). Some bases may take both nominal and verbal inflection (possibly with slightly different [though related] meanings depending on grammatical category). An example of such a base is *imiq-*, which means “drinking water; an alcoholic drink” when followed by nominal morphology, and “to drink” when followed by verbal morphology (MacLean 1981:14); another example is *auk-*, which means “blood” as a noun and “to melt; to bleed” as a verb (MacLean 1981:9). There are three possible treatments for such bases (or base pairs). The first is to treat them as a single lexeme with membership in multiple categories. The second is to treat them as a single lexeme belonging to a single category and recognize a zero derivational suffix transforming them into members of a second category. The third is to treat them as separate lexemes, though obviously descended from the same historical lexeme and with

¹⁶This term is borrowed from Lowe (1996). I prefer it over “morpheme” because many formatives may be analyzed as multimorphemic.

clear semantic overlap. The first treatment fails to explain semantic differences between the two versions of the base (beyond the fact that one is nominal and the other verbal). The second inherits this flaw and additionally requires a way to specify which bases accept a zero derivational suffix, since many or most bases clearly belong to only one category. What the third treatment lacks in terms of explicitly linking the two bases it makes up for in its ability to differentiate between them on semantic as well as categorical grounds. This is the approach I adopt in the present work.

Nominal bases can be grouped as “noun bases proper” (hereafter simply “noun bases”), personal pronoun bases, and demonstrative bases, depending on the inflectional paradigm that pertains to a given base. Noun bases proper cannot be subclassified in the same way that verb bases can, but a group of noun bases called “positional nouns” probably deserve their own subcategory. These are morphologically noun bases which serve a similar function to prepositions in English, expressing relative physical or metaphorical position. These bases are generally followed by either possessive inflection or special positional noun postbases (Seiler 2005:22). Put another way, they take noun inflectional endings, but there is a large set of noun suffixes that they cannot (or generally do not) take, as well as a small set of suffixes they can take, which other noun bases cannot. Examples of positional noun bases include *akî* ‘area opposite’, *sivu* ‘area in front; time prior’, and *sani* ‘area by the side’ (see examples (42)–(44)). Postbase *-nmuk-* ‘go/move toward ____’ in example (44) is a suffix which normally attaches to demonstratives but can also attach to positional noun bases (but not other noun bases, which take *-muk-* instead).

Ex. (42) *akiptinni* *igluqaqtut*
 akî-ptinni *iglu-qaq-tut*
 opposite-LOC.SG.1PL house-have-IND.PRS.3PL
 ‘They have a house across from ours’ (MacLean n.d.a)

Ex. (43) *sivuani* *tikitchuani*
 sivu-ani *tikî-tuani*
 time.before-LOC.SG.3SG arrive-textscind.pst.1pl
 ‘We arrived before she/he did’ (MacLean n.d.a)

Ex. (44) *saniñmugun* *taamna*
 sani-nmuk-un *ta(t)-am-na*
 area.by.side-move.toward-IMP.2SG.3SGO near.listener-that.one.RESTR-ABS.SG
 ‘Turn that one sideways’ (MacLean 1986b:53)

Postbases may be classified in a manner similar to bases, but whereas bases are only categorized along one axis (the class(es) of inflectional endings that may be suffixed to them), postbases are categorized along two: the class(es) of inflectional endings that may be suffixed to them (in other words, the category they derive), and the class(es) of bases to which they may be suffixed (the category from which they derive) (see de Reuse 1992:164; Smith 1980:283–284). For postbases which derive verbs, a distinction must be made between those which alter valence (causatives and passives, for example) and those which do not (Fortescue 2004:1395, Mithun 2000).

While bases, postbases, and inflectional endings are the most obvious categories of formatives, several

TABLE 2.5: “Post-inflectional” derivational suffixes

Suffix	Attaches to	Results in
-aq-	Demonstratives in vialis case	Verb expressing travel through or by way of a place or area (see example (47) on the preceding page)
-q-	Demonstratives in ablative or terminalis case	Verb expressing travel to (for demonstratives in terminalis) or from (ablative) a place or area (see example (48) on the preceding page)
-nmun	Demonstratives in terminalis case	Nominal expressing direction of movement or progression of an event toward a place or area (see example (49) on the preceding page)
-nmuk-	Demonstratives in terminalis case	Verb expressing direction of movement or progression of an event toward a place or area (see example (50) on the following page)
-miñ	Demonstrative adverbs which denote an extended area and are in ablative case	Nominal expressing origin of movement or progression of an event away from an area (see example (51) on the next page)
-mik-	Demonstrative adverbs which denote an extended area and are in ablative case	Verb expressing origin of movement or progression of an event away from an area (see example (52))
-[m/ɣn/n]iit-	Nouns; suffix is derived from locative case ending and reflects the grammatical number of the noun stem to which it attaches	Verb expressing location (see example (8) on page 12 and example (53) on the next page)
-[m/ɣn/n]juk-	Nouns; suffix is derived from terminalis case ending and reflects the grammatical number of the noun stem to which it attaches	Verb expressing travel to a place (see example (54) on the following page)
-[m/ɣn/n]iñɣaq-	Nouns; suffix is derived from ablative case ending and reflects the grammatical number of the noun stem to which it attaches	Verb expressing arrival from a place (see example (55) on page 33)
-[k/k/tig]jaaq-	Nouns; suffix is derived from vialis case ending and reflects the grammatical number of the noun stem to which it attaches	Verb expressing travel through or by way of a place (see example (56) on page 33)
-aglaaq-	Nouns in terminalis case	Verb expressing arrival as far as a point in space or time (see example (57) on page 33)
-qsiuq-	Nouns, demonstrative pronouns, and demonstrative adverbs in locative case	Verb expressing experience or activity in or near a place, area, or object (see example (58) on page 33)

- Ex. (50) *pauᅇanmuktuak* *uvlaaq uniāgaqlutik*
pag-uᅇa-nmuk-tuak *uvlaaq uniāgaq-vlutik*
 landward.EXT-TRM-go.toward-IND.PST.3DU today travel.by.dogsled-CONTEMP1.3DU
 ‘They [two] were heading inland this morning by dogsled.’ (MacLean 1986b:51)
- Ex. (51) *avaᅇᅇamiñ* *tusaagikput* *nipattuaq*
av-aᅇᅇa-miñ *tusaa-kikput* *nipat-tuaq*
 down.the.coast.EXT-ABL-origin hear-IND.PRS.1PL>3SGO make.sound-PART.3SG.ABS
 ‘We heard the sound originating from down the coast.’ (MacLean 1986b:51)
- Ex. (52) *avaᅇᅇamiksut*
av-aᅇᅇa-mik-tut
 over.there/down.coast.some.distance.from.shore.EXT-ABL-moving.from-IND.PRS.3SG
 ‘They are on their way up the coast from down the coast.’ (MacLean 1986b:52)
- Ex. (53) a. *aᅇnaᅇniiniaqtuᅇa*
aᅇnaq-ᅇniit-niaq-tuᅇa
 woman-be.located.DU-FUT-IND.PRS.1SG
 Alternatively:
aᅇnaq-ᅇni-it-niaq-tuᅇa
 woman-LOC.DU-be-FUT-IND.PRS.1SG
 ‘I will be with the [two] women’ (MacLean 1986a:118)
- b. *aᅇnaniinniaqtuᅇa*
aᅇnaq-niit-niaq-tuᅇa
 woman-be.located.PL-FUT-IND.PRS.1SG
 Alternatively:
aᅇnaq-ni-it-niaq-tuᅇa
 woman-LOC.PL-be-FUT-IND.PRS.1SG
 ‘I will be with the [three or more] women’ (MacLean 1986a:118)
- Ex. (54) a. *iglumuktuaq*
iglu-muk-tuaq
 house-go.to.SG-IND.PST.3SG
 Alternatively:
iglu-mun-k-tuaq
 house-TRM.SG-go-IND.PST.3SG
 ‘He went to the house’ (MacLean 1986a:118)

- b. *aġnanugniaqtuḡa*
aġnaq-nuk-niaq-tuḡa
 woman-go.to.PL-FUT-IND.PRS.1SG
 Alternatively:
aġnaq-nun-k-niaq-tuḡa
 woman-TRM.PL-go-FUT-IND.PRS.1SG
 ‘I will go to the women’ (MacLean 1986a:118)
- Ex. (55) *sumiñḡaqpisi?*
su-miñḡaq-pisi
 what-come.from-INT.2PL
 Alternatively:
su-miñ-ḡaq-pisi
 what-ABL.SG-come.from-INT.2PL
 ‘Where did [all of] you arrive from?’ (MacLean 1986a:119)
- Ex. (56) a. *naḡirvikuaqpisik* *uvlupak*
naḡirvīk-kuaq-pisik *uvlupak*
 hospital-go.by.way.of.SG-INT.2DU today
 Alternatively:
naḡirvīk-kun-aq-pisik
 hospital-via.SG-go-INT.2DU
 ‘Did you [two] go by way of the hospital today?’ (MacLean 1986a:190)
- b. *iglutiguaqtuat* *qiñiġivuk*
iglu-tiguaq-tuat *qiñiġ-kivuk*
 house-go.through.PL-PART.3PL.ABS watch-IND.PRS.1DU>3DU/PL
 Alternatively:
iglutigun-aq-tuat
 house-via.PL-go-PART.3PL.ABS
 ‘We are watching the ones that are going through the houses.’ (MacLean 1986a:190)
- Ex. (57) a. *qitqanuaglaaġami* *utiġmiuq*
qitiġ-nun-aglaaq-kamī *utiq-mmī-tuq*
 middle-TRM.SG.3SG-arrive.as.far.as-CONSEQ.3RSG return-also-IND.PRS.3SG
 ‘When she got halfway through it, she returned unexpectedly.’ (MacLean n.d.a)
- b. *sumuaglaaqpat* *unnuaq*
su-mun-aglaaq-pat *unnuaq*
 what-TRM.SG-arrive.as.far.as-INT.3PL last.night
 ‘How far did they progress last night?’ (MacLean n.d.a)

- Ex. (58) a. *umiamiqsiugnasi*
umiaq-mi-qsiuq-nasi
 boat-LOC.SG-experience-NEGCONTEMP.2PL
 ‘Don’t hang around the boat.’ (MacLean n.d.a)
- b. *samaniqsiuqpisi*
samma-ni-qsiuq-pisi
 down.there.NV-LOC-experience-INT.2PL
 ‘Have you been down there?’ (MacLean n.d.a)

There is a certain amount of inconsistency in the list in Table 2.5; most of the suffixes which attach to nouns are presented as postbases derived from an inflectional suffix followed by a derivational suffix, while *-aglaaq-*, *-qsiuq-* and suffixes which attach to demonstratives are listed detached from the inflectional endings with which they co-occur. The suffixes are given in the table as they appear in MacLean (1986a), MacLean (1986b), and, in the case of *-aglaaq-* and *-qsiuq-*, MacLean (n.d.a). Examples (53) through (56) present two analyses for the nominal suffixes: as postbases containing inflectional information, and as formatives divisible into inflectional and derivational components (with the derivation coming second). Smith (1980:282–283) provides yet another analysis for a similar phenomenon in Labrador Inuttut: that the formatives I have been calling inflectional endings be considered postbases when followed by non-enclitic derivational morphology. However one chooses to analyze them, the fact that these suffixes either exhibit paradigmatic variation or attach to inflectional endings (or, in Smith’s analysis, postbases that would otherwise be considered inflectional) differentiates them from the vast majority of postbases. For computational morphology, the uniqueness of these postbases requires unique treatment: if they are viewed as compounds containing inflectional information, this inflectional information must be conveyed in analyses; if viewed as suffixes to inflectional endings, they must be specified with special morphotactic rules from which most postbases would be exempt.¹⁹

The remaining two categories of formatives—enclitics and reduced (cliticized) words—differ from those discussed so far in that, while they are phonologically and orthographically bound to host words, they can be considered syntactic words in their own right (Miyaoaka 2000:228; Woodbury 2002:84; de Reuse 1992:163). Reduced words differ from enclitics in that they correspond to full forms which can exist prosodically separate from a host word, whereas enclitics cannot. Both enclitics and reduced words attach at the extreme right of the prosodic word, and many linguists may prefer to treat them as a single category (see for example Woodbury 2002:89–91). Certainly, the reduced forms are as much enclitics as *-n’t* in English ‘don’t.’ The decision to distinguish between them for purposes of the Iñupiaq transducer arose from a March 2009 discussion between Edna MacLean, Larry Kaplan, Lori Levin, Bob Frederking, Eliot DeGolia, Ida Mayer, and myself, where we concluded that reduced words are fundamentally different from Iñupiaq enclitics (just as English *-n’t* is fundamentally different from possessive *-’s*), and that making a formal distinction may be beneficial to language learners in particular.

Some important enclitics include *=gguuq* (reportative evidential), *=li* (indicates change of subject), *=lu* ‘and’, and *=tuq* ‘I hope’. These are illustrated in examples (59)–(62) below.

¹⁹See also Lowe (1996:222–223) for examples of the Inuktitut post-inflectional suffix *-uq* ‘act, behave’ following terminalis, vialis, and similaris noun inflectional endings; this suffix has no direct equivalent in North Slope Iñupiaq (Lawrence Kaplan, personal communication, 7 May 2009) but demonstrates that the phenomenon of post-inflectional derivation extends beyond the North Slope.

- Ex. (59) *umiagguuq unna atuǵniaǵaa*
umiaq=gguuq unna atuq-niaq-kaa
 boat.ABS.SG=REP down.there.EXT use-FUT-IND.PRS.3SG>3SGO
 ‘He said he will use that boat which is down there’ (MacLean 1986a:44)
- Ex. (60) *aǵnaq miquqtuq, aǵunli siñiktuq*
aǵnaq miquq-tuq aǵun=li siñik-tuq
 woman.ABS.SG sew-IND.PRS.3SG man.ABS.SG=change.of.subject sleep-IND.PRS.3SG
 ‘The woman is sewing, whereas the man is sleeping.’ (MacLean 1986a:123)
- Ex. (61) *amaǵuǵlu qimmiǵlu paǵaliktuk*
amaǵuq=lu qimmiq=lu paǵalik-tuk
 wolf.ABS.SG=and dog.ABS.SG=and run(of.animal)-IND.PRS.3DU
 ‘The wolf and the dog are running’ (MacLean 1986a:34)
- Ex. (62) *īlisaǵilagutuq*
īlisaǵi-lagu=tuq
 recognize-OPR.1SG>3SGO=I.hope
 ‘I hope I recognize him’ (MacLean 1986a:44)

Reduced words may be particles, personal pronouns, or demonstratives. Examples are given below:

- Ex. (63) *Sunauna?*
suna=uv-na
 what.ABS.SG=here.RESTR-ABS.SG
 ‘What’s this?’
- Ex. (64) *Aquvittuaq iglarraqsiv!uniasii*
aquvit-tuaq iglaq-rraqsî-vlunî=aasî
 sit-IND.PST.3SG laugh-begin-CONTEMP1.REAL.3SG=and.then
 ‘He sat down and began to laugh’ (MacLean 1986b:10)

Enclitics and reduced words can appear in the same phonological/orthographic word, as in *taimmagu-uquna* in example (65) and *iñuqaǵnai!aummiv!uniluuuvva* in example (66).

- Ex. (65) *Taimmaguuquna iñuuniagñiqsuq aahaaliq*
taimma=guuq=una iñuuniaq-nîq-tuq aahaaliq
 once=REP=this live-reportedly-3SG longtailed.duck
piyaaǵi!u tapqami
piyaaq-ǵî=lu tapqaq-mî
 young.animal-ABS.PL.3SG small.island/sand.spit-LOC.SG
 ‘Once, a long-tailed duck lived with her ducklings on a small island/sand spit, it is said.’

(Kaveolook 1974:1)

- Ex. (66) *iñuqaǵnaiļaummiṽluniļuvva*
iñuqaǵnaiļaq-u-mmi-ṽluni=lu=uvva
 murderer-COP-also-CONTEMP1.REAL.3SG=and=here
 ‘and he also being the murderer here’ (Nashaknik 1973:34)

2.1.4.3 Morphotactic Constraints

In the literature on Eskimo languages, the construction of a phonological or orthographic word is often explained with a schema such as the following:

word = base + zero or more postbases + inflection + zero or more enclitics

(see for example de Reuse 1992:163; Smith 1980:282; Lowe 2000:154; Miyaoka 2000:228). A few scholars have proposed more detailed morphotactic models. Fortescue’s (1980) treatment of West Greenlandic formative ordering is based on a subcategorization of postbases which is primarily motivated by semantic considerations. While the model is relatively complex, only one of his rules actually makes any purely morphotactic predictions beyond those in the formula above.²⁰ The postbases for which Fortescue’s model predicts a specific order are what he calls “sentential verbal suffixes,” which he understands to have clause-level rather than word-level scope (Fortescue 2004:1395); these include the categories of **tense** (Seiler [1997] prefers to call this aspect; examples from West Greenlandic include *-qqammir-* “recently,” *ssa* “future”), **epistemic modality** (for example, *-gunar-* “it seems/probably,” *-ssagaluar-* “should/would—irrealis”), **negation**²¹ (*-nngit-* “not,” *-pianngit-* “not really”), **subjective/narrative coloration** (*-ratar-* “at last/surprise; *-kasig-/kassag-* “disdain/complicity/dear old”), and **conjunction** (*-galuar-* “although,” *-qqajanngit-* “long before”) (all examples from Fortescue 1980:277). Fortescue’s proposed ordering rule for these (1980:261) is reproduced in Figure 2.3. All elements in this rule are optional, but they must occur in the order specified, and no other postbases may follow.

$$(V_{ten}) (+ V_{ep}) \left\{ \begin{array}{l} (+ V_{neg}) (+ V_{sub}) \\ (+ V_{conj}) \end{array} \right\}$$

FIGURE 2.3: Fortescue’s sentential verbal suffix ordering rule

Seiler (1997) has proposed an extension to Fortescue’s model, but like most of Fortescue’s model itself it predicts only the semantic scope of postbases, rather than specifying purely morphotactic constraints

²⁰What his rules do predict that those inherent in the basic schema do not is constituency and semantic scope, but these predictions, while extremely important, fall outside the scope of this thesis.

²¹Other rules also include negative suffixes; in those rules, the negative suffixes are not considered sentential verbal suffixes, and no meaningful constraints on their possible position relative to other non-sentential suffixes are expressed.

(that is, constraints which would apply regardless of semantics, such as the constraint that deverbal postbases attach to verb stems). de Reuse (1988) has proposed an alternative model, based on his work with Siberian Yupik, which makes different, though not necessarily more, morphotactic predictions. The model, reproduced in Figure 2.4, distinguishes between “absolutely ordered” and “relatively ordered postbases; zero or more relatively ordered postbases may occupy any slot marked with a row of dots. Absolutely ordered postbases, which correspond roughly to Fortescue’s sentential postbases, are also optional but may only occupy specifically designated slots. There is a fundamental disagreement between the two models: Fortescue’s does not allow any interleaving of sentential and non-sentential postbases, while de Reuse’s allows extensive interleaving, except between slots 4 and 5.

Verb stem;

 1. *MODALITY;*

 2. *PAST TENSE;*

 3. *PROGRESSIVE;*

 4. *FUTURE/INEFFECTIVE;*
 5. *EVIDENTIALS;*
Verb inflection.

FIGURE 2.4: De Reuse’s “position-based” morphotactic model

To my knowledge, no serious attempt has been made to classify Iñupiaq postbases according to categories such as Fortescue’s and de Reuse’s. Without such a classification, it is impossible to evaluate how well these models apply to Iñupiaq.

2.2 Computational Morphology

Morphological analysis (or something akin to it) is essential to fundamental NLP tasks such as identifying a word’s lemma, grammatical category, inflection, etc. Most computational techniques for obtaining morphological (or morphology-like) information can be assigned to one of four categories: dictionary lookup, stemming, finite-state morphology, or machine learning. Although machine learning promises to be the “next big thing” in computational morphology, I omit it from the discussion that follows. An overview can be found in (Roark and Sproat 2007:116–136), who note that “despite the substantial progress in recent years on automatic induction of morphology, there are also still substantial limitations on what current systems are able to handle” (p. 136); in contrast, many hand-written analyzers are able to handle “a significant portion of the morphology of even a morphologically complex language such as Finnish, within the scope of a doctoral-dissertation-sized research project” (Roark and Sproat 2007:116–117; see also Beesley 2004c).

In the rest of the section, I will discuss the strengths and weaknesses of the dictionary lookup, stem-

ming, and finite-state approaches to computational morphological analysis.

2.2.1 Dictionary Lookup

Particularly in the case of morphologically simple languages such as English, some applications can satisfy their morphological analysis requirements with the aid of a “dictionary”—a table or database whose keys are fully derived, fully inflected words (Sproat 1992:xi; Anderson 1988:1). Advances in the processing power and storage capacity of computer hardware make dictionary lookup approaches increasingly feasible (Sproat 1992:xii; Anderson 1988:2), perhaps even for morphologically complex languages like Finnish (Church 2005). Church (2005) argues that the more complex the methods used to do computational morphology, the greater the risk of errors in general, and the greater the risk of more dangerous errors than would be introduced by simpler methods. On these grounds, he advocates dictionary lookup methods where feasible (except in cases where it is reasonable to avoid morphological analysis altogether).

Frequently, however, dictionary lookup approaches are insufficient, unfeasible, or undesirable. Anderson (1988:2) and Sproat (1992:xii–xiii) point out that an approach in which each word is listed separately fails to capture the redundancy of the lexicon (i.e., that the same morphemes appear in multiple words) and is thus both inefficient and descriptively deficient. Sproat (1992:xii–xiii) raises perhaps a more important criticism: that “no dictionary contains all the words one is likely to find in real input.” This is especially true for morphologically rich languages.

2.2.2 Stemming

In the field of information retrieval, documents are indexed according to the terms they contain. It is often beneficial to treat the various inflected and derived forms of a given lemma as instances of the same term. The process of mapping these variants onto some canonical form is known as term conflation or word normalization and is often accomplished using one of several techniques collectively known as “stemming” algorithms.

Following Galvez et al. (2005), I will draw a distinction between stemmers and lemmatizers, both of which are used for term conflation. Lemmatizers are considered “linguistic” methods because they identify lemmas through morphological analysis, including full consideration of morphographemic processes and morphotactic constraints. The lemmas they identify are generally “real” orthographic words. Lemmatizers are generally implemented using finite-state morphology (although there are exceptions; see for example Khaltar and Fujii [2008]); “linguistic” finite-state morphology will be discussed in Section 2.2.3 on pages 39–50.

Although stemming algorithms are usually motivated by linguistic analysis of the language to which they apply, they are considered “non-linguistic” methods in that they do not actually analyze the morphology of the words they operate on and consequently identify “stems” which may not be linguistically meaningful (Galvez et al. 2005:523; Pirkola 2001:332–333) and which occasionally conflate terms improperly, or mistakenly index terms which should be ignored. For example, the Porter stemmer (Porter 1980), one of the most popular stemming algorithms for English, conflates “general”, “generous”, and “generation” under the stem *gener*, but fails to conflate “recognize” and “recognition”, assigning these to the stems *recogn* and *recognit*, respectively (Hull and Grefenstette 1996:2). Krovetz (1993:192) points out

that the Porter stemmer conflates “doing” to *doe*; as a result, the word is likely to be indexed, even though the actual lemma, “do”, is generally considered unimportant for information retrieval (see also Pirkola 2001:334-335).

Given that the *raison d’être* of stemmers is to conflate terms, it should come as no surprise that (even ignoring conflation errors) stemmer output generally makes poor input for further analysis, such as syntactic parsing (Galvez et al. 2005:523) or word sense disambiguation (Krovetz 1993:192; note that Krovetz’s stemmer was designed in an attempt to address this shortcoming). What may be surprising is that there is a debate over whether term conflation generally improves information retrieval results, especially for morphologically impoverished languages like English (Manning and Schütze 1999:132-133). Harman (1991) found that stemming made no significant difference in English retrieval performance. However, Krovetz (1993) and Hull and Grefenstette (1996) conclude that term conflation is consistently, if modestly, beneficial in English information retrieval. In many languages with more complex morphology, term conflation has been more demonstrably effective; these include French (Savoy 1999), Swedish (Carlberger et al. 2001), Finnish (Korenius et al. 2004; Kettunen et al. 2005), and Turkish (Ekmekçioğlu and Willett 2000; Sever and Bitirim 2003). Pirkola (2001) makes two typology-based predictions about the utility of term conflation: first, that conflation is more beneficial in languages with more complex inflectional systems (see also Manning and Schütze 1999:133), and second, that conflation of derived forms and/or compounds is more useful the more transparent a language’s derivational/compounding semantics are.

Hull and Grefenstette (1996), Kettunen et al. (2005), Korenius et al. (2004), and Galvez and Moya-Anegón (2006) all show that lemmatization can be as effective as (in the case of the last two articles, more effective than) non-linguistic stemming approaches at conflating terms. The main drawback of lemmatizers is their inability to deal with unknown words, but as Koskenniemi (1996) demonstrates, this can be overcome by including a “guessing” component which can analyze the derivational and/or inflectional morphology of a word, even if the stem is unknown (see Section 2.3.7 on pages 59-60). Another frequently-cited drawback, slowness (Galvez et al. 2005:528; Alegria et al. 1996:198; Sever and Bitirim 2003), is probably more a property of the particular method used than of lemmatization itself; for example, Alegria et al. (1996) report a speed increase on the order of 1000 times by switching from a KIMMO-style system (see pages 45-48) to a lexical transducer (see pages 48-50). A final drawback which may be inevitable is that creating a lemmatizer (a full-fledged morphological model of a language) typically requires more time, more linguistic knowledge, and a more thorough development process than creating a passable stemmer. Despite the advantages of lemmatization over stemming (including increased accuracy (Koskenniemi 1996; Galvez et al. 2005; Galvez and Moya-Anegón 2006) and linguistically meaningful results usable in other NLP analyses (Galvez et al. 2005)) it is probably due to this last drawback that stemming remains popular in information retrieval.

2.2.3 Finite-State Morphology

For morphologies of any complexity, full morphological analysis requires phonological analysis.²² Since Koskenniemi’s dissertation was published in 1983, finite-state machines have gained wide acceptance in

²²Since most morphological transducers operate on orthographic rather than phonetic forms, it would be more accurate to speak of graphemics than phonology; however, for the sake of simplicity, I will continue to lump graphemics under the umbrella of phonology in this section.

computational models of phonology and morphology (Roark and Sproat 2007:101). Their simplicity gives rise to important mathematical properties which make them compact, efficient, and scalable. Beesley (2004c:3) writes, “Finite-state methods certainly cannot do everything in natural language processing; but when they are appropriate, they are extremely attractive.” The two main approaches to finite-state morphology have been two-level systems and lexical transducers.

2.2.3.1 Definitions

Informally, a finite-state automaton (FSA) is a directed graph that represents or “accepts” a set of strings of symbols called a (formal) “language”. Figure 2.5 shows a simple FSA that represents the language {can, cast, cost, man, mast, most}. It is made up of *states* (the numbered circles) connected by *transitions* (shown as labeled arrows; in graph theory, these are called *edges*). State 1 is the *initial state*, and state 6 is the *final* or *accepting state*. The language accepted by an FSA is the set of strings that can be constructed by concatenating all the labels along some path of edges which begins at the initial state and ends at some final state (an FSA may have multiple final states, but only one initial state).

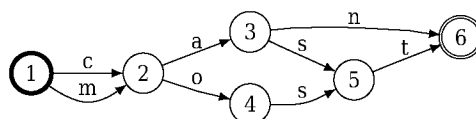


FIGURE 2.5: An FSA that computes the regular language {can, cast, cost, man, mast, most}

While the number of states and transitions in an FSA must be finite, as the name suggests, the language represented by an FSA may be infinite.²³ Figure 2.6 on the following page illustrates one such language: the set of all strings conforming to a series of constraints I will call “simplified Hawaiian phonotactics.” The FSA accepts any string, regardless of length, which consists of symbols from the Hawaiian alphabet, ends in a vowel, and contains no consonant clusters. The comma-separated lists in the figure are equivalent to separate edges for each item in the list.

Like an FSA, an FST is a directed graph, but rather than simply representing a set of strings of symbols, it maps sets of strings of symbols onto other sets of strings of symbols; that is, it expresses a *relation* between a given number of formal languages. The number of formal languages in a relation is called the *arity* of the relation and may be arbitrarily large (Kaplan and Kay 1994:340), though FSTs modeling linguistic phenomena rarely have an arity greater than two (though see Kiraz 2000, 2001). An FSA can be considered an FST with an arity of one, or in other words an FST which computes a unary relation; it

²³However, not all infinite languages can be represented by FSAs. For example, no FSA can represent a language with a constraint of the type $a^n b^n$ where n is unbounded. So an FSA cannot be used to test, for example, whether the parentheses in a string are balanced, unless one places a finite upper limit on allowable parenthesis nesting.

There is a scale in formal language theory called the Chomsky(-Schützenberger) hierarchy which classifies formal grammars by restriction. The most restricted category is regular grammars; these are followed by context-free, context-sensitive, and unrestricted grammars. $a^n b^n$ is context-free in power, and FSAs can only compute regular languages, so no FSA can handle a language with such a constraint.

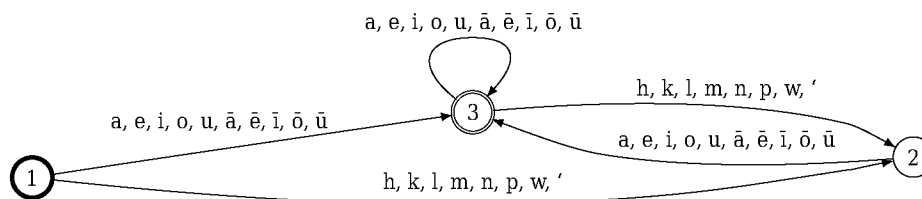
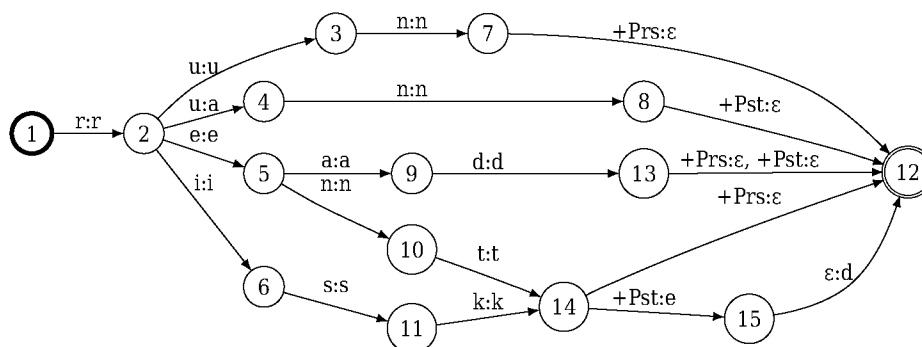


FIGURE 2.6: An FSA modeling simplified Hawaiian phonotactics

follows that all special properties of FSTs which do not derive from an FST's arity are shared by FSAs, and vice versa.

An FST with an arity of two is said to have an *upper language* and a *lower language*. Figure 2.7 shows a simple FST which computes the present (non-3sg) and past tense forms of the words *run*, *read*, *rent*, and *risk*. The label on each edge has two parts, separated by a colon; the left part belongs to the upper language, the right part to the lower language. The symbol ϵ indicates an empty string. The lower language of this FST is the set of surface forms {run, ran, read, rent, rented, risk, risked}; the upper language contains the citation form of each word, suffixed with the grammatical tags *+Prs* (present) and *+Pst* (past). Most of the correspondences represented in this particular FST are one to one (for example, *risked* in the upper language corresponds only to *risk+Pst* in the lower language, and vice versa), but there is one one-to-many correspondence: *read* in the upper language corresponds to both *read+Prs* and *read+Pst* in the lower language. Although the FST in Figure 2.7 contains no many-to-one or many-to-many correspondences, these may also be represented in an FST.

FIGURE 2.7: An FST that computes present (non-3sg) and past forms of the words *run*, *read*, *rent*, and *risk*

Just as an FSA can represent a set of all strings corresponding to an abstract pattern (as in the simplified Hawaiian example in Figure 2.6), an FST can represent a relation between all strings which correspond according to a particular pattern (as long as the pattern is regular in power; see footnote 23 on the preceding page). A phonological rewrite rule can be thought of as a pattern of correspondences. Figure 2.8 gives

an example of a phonological rule ($n \rightarrow \tilde{n} / \hat{i}$ —, part of the palatalization process in Iñupiaq; see 2.1.2.2 on pages 9–11) encoded as a finite-state transducer. Σ denotes the alphabet or set of symbols recognized by the transducer; it contains \hat{i} , n , \tilde{n} , and any other symbol that might appear as input to the transducer. σ is a wildcard denoting any symbol in Σ . In this transducer, the upper language is the set of all strings which can be built from symbols in Σ , and the lower language is the set of all strings which can be built from symbols in Σ and which do not contain the sequence $\hat{i}n$. The upper language represents an abstract, “underlying” form, and the lower language represents the corresponding “surface” form.

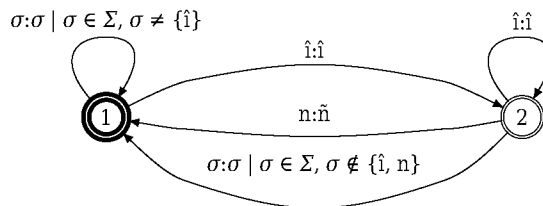


FIGURE 2.8: AN FST that computes the regular relation corresponding to the rule $n \rightarrow \tilde{n} / \hat{i}$ —

The transducer begins in state 1, and transitions back to the same state unless it encounters the symbol \hat{i} , in which case it transitions to state 2. From state 2, the transducer transitions back to state 1 on any input other than \hat{i} ; as long as the input is \hat{i} , the transducer stays in state 2. For each symbol in the input, the transducer outputs a symbol. In this transducer, the output symbol is identical to the input symbol unless the transducer is transitioning from state 2 to state 1 on \tilde{n} (if it is being applied upward, that is, with lower language input and upper language output) or n (in either direction). If the transducer is applied downward and receives the input symbol n while in state 2, it will emit the symbol \tilde{n} . If the transducer is applied upward and receives the input symbol \tilde{n} while in state 2, there are two possible transitions: \tilde{n} may correspond to n , or to \tilde{n} (by the transition $\sigma:\sigma \mid \sigma \in \Sigma, \sigma \notin \{\hat{i}, n\}$); this is because the upper language is unrestricted and thus allows the sequence $\hat{i}\tilde{n}$ as well as $\hat{i}n$. Because there are two possible corresponding symbols, the transducer emits two strings. Thus, if the transducer is applied upward on the input string $\hat{i}\tilde{n}uk$, it generates two results, $inuk$ and $\hat{i}\tilde{n}uk$; if the transducer is applied downward on either $inuk$ or $\hat{i}\tilde{n}uk$, it generates $\hat{i}\tilde{n}uk$. Put a different way, one of the relations encoded by the transducer is the many-to-one relation $(\{inuk, \hat{i}\tilde{n}uk\}, \hat{i}\tilde{n}uk)$. If the transducer is applied upward and receives the input symbol n while in state 2, there is no possible transition; the sequence $\hat{i}n$ is disallowed by the lower language. The transducer would fail on this input.

2.2.3.2 Mathematical Properties of Finite-State Machines

FSAs and FSTs provide several advantages over other possible computational approaches to morphology. Many of these advantages stem from the fact that FSTs compute regular (or rational) relations (Roark and Sproat [2007:5] Roche and Schabes [1997:14]; unary regular relations, which FSAs compute, are called regular languages [Roark and Sproat 2007:4; Roche and Schabes 1997:5]).

Not all linguistic phenomena can be modeled by regular languages or relations; syntax in particular has gained notoriety as a system not describable by a regular grammar, thanks to Chomsky (1956). However, phonological rules, at least within the framework of Generative Phonology, are constrained enough to be describable by regular relations (Johnson 1972; Kaplan and Kay 1994). Morphological operations, including affixation, root-and-pattern morphology, and (to some extent) reduplication, can also be described by regular languages (see Roark and Sproat 2007:23–61).

One of the most useful attributes of regular languages and relations is their closure properties, which are enumerated in Table 2.6 on the next page (Kaplan and Kay 1994:340–344; Roche and Schabes 1997:5–6, 17–19; Beesley and Karttunen 2003:54–56).²⁴ These properties define the operations that can be used to derive complex regular languages and relations from one or more simpler regular languages or relations. A set is said to be closed under a given operation if, when one applies the operation to members of the set, the result is a member of the set. For example, the set of regular languages is closed under intersection, so if L_1 and L_2 are regular languages, then the intersection of L_1 and L_2 is also a regular language. Likewise, regular relations are closed under composition, so if R_1 and R_2 are regular relations, then the composition of R_1 and R_2 is also a regular relation. Additionally, the identity relation of a regular language is a regular relation, and the Cartesian product of two regular languages is a regular relation.

When a complex phenomenon can be modeled by a regular language or relation, and the phenomenon

²⁴Following are brief explanations for the operations in Table 2.6:

Concatenation ($L_1 \cdot L_2$) joins two languages or relations in sequence. If L_1 is {fast, small} and L_2 is {er, est}, then $L_1 \cdot L_2$ is {faster, smaller, fastest, smallest}.

Kleene closure (L^*) licenses zero or more repetitions of a language or relation. If L_1 is the regular language {a}, then L_1^* licenses the strings “a”, “aa”, “aaa”, “aaaa”, etc., as well as the empty string (zero instances of ‘a’).

Union ($L_1 \cup L_2$) creates a language or relation containing all members of each of its operands. If L_1 is {big, large} and L_2 is {huge, gigantic}, then $L_1 \cup L_2$ is {big, large, huge, gigantic}.

Reversal (L^- or L^R) produces a language or relation that is the reverse of its operand. If L is {edit, keel, drawer, relaid}, then L^- is {tide, leek, reward, dialer}.

Intersection ($L_1 \cap L_2$) creates a language containing only those members of L_1 which are also members of L_2 . If L_1 is {car, truck, bicycle} and L_2 is {car, card, carry}, then $L_1 \cap L_2$ is {car}.

Difference or Relative Complement ($L_1 - L_2$ or $L_1 \setminus L_2$) creates a language containing all members of L_1 which are not members of L_2 . If L_1 is {car, truck, bicycle} and L_2 is {car, card, carry}, then $L_1 - L_2$ is {truck, bicycle}.

Complementation or Absolute Complement (\bar{L}) produces a language containing all strings over an alphabet Σ which are not members of L . If Σ corresponds to the Roman alphabet and L is {apple, pear, peach}, then \bar{L} is a language containing all strings of Roman alphabet letters (including “apples,” “peachy,” “pearl,” “banana,” “bzfqlllmrjpcqiffzvw,” etc.) except “apple,” “pear,” and “peach”.

Composition ($R_1 \circ R_2$) creates a relation that maps strings from the upper language of R_1 directly onto the lower language of R_2 . If R_1 is the relation {(one,un), (two,deux), (three,trois)} and R_2 is the relation {(un,ett), (deux,två), (trois, tre)}, then $R_1 \circ R_2$ is the relation {(one,ett), (two,två), (three,tre)}. So, by composing R_1 and R_2 , we can compute the Swedish equivalent for a given English number without recourse to any intermediate value (the equivalent French number). Composition of relations is analogous to the algebraic concept of composition of functions, but the notational convention for functional composition is the reverse of the convention for relational composition (Matoušek and Nešetřil 2009:25); that is, for relations, the part to be computed first is listed first, but for functions, the part to be computed first is listed last: if $f(x) = 2x + 1$ and $g(x) = 3x$ then $g \circ f = g(f(x)) = (3(2x + 1)) = 6x + 3$.

Inversion (R^{-1}) creates a relation identical to its operand except that the upper and lower languages are switched. If R is {(red,rouge), (green,vert), (blue,bleu)}, then R^{-1} is {(rouge,red), (vert,green), (bleu,blue)}.

Identity ($Id(L)$) creates a relation whose upper and lower languages are both L . If L is the regular language {me, myself, I}, then $Id(L)$ is {(me,me), (myself,myself), (I,I)}. The identity relation of an FSA may be combined with an FST using operations such as union, composition, etc.

Cartesian product (sometimes called **cross product**) ($L_1 \times L_2$) creates a relation such that each element in L_1 is associated with each element in L_2 . If L_1 is {astute, diminutive, shy} and L_2 is {mouse, cat, giraffe} then $L_1 \times L_2$ is {(astute,mouse), (diminutive,mouse), (shy,mouse), (astute,cat), (diminutive,cat), (shy,cat), (astute,giraffe), (diminutive,giraffe), (shy,giraffe)}.

TABLE 2.6: Closure of regular languages and relations under various operations

Operation	Languages	Relations
concatenation	yes	yes
Kleene closure	yes	yes
union	yes	yes
reversal	yes	yes
intersection	yes	no
difference	yes	no
complementation	yes	no
composition	n/a	yes
inversion	n/a	yes

can be broken down into smaller constituent phenomena which can also be modeled by regular languages or relations, then the closure properties of regular languages and relations provide an elegant and mathematically straightforward way to construct the model. For example, one can model the phonology of a (natural) language in a single transducer by creating individual transducers for each phonological rule, and then composing these together in the order in which they should fire (Johnson 1972; Kaplan and Kay 1994); this is conceptually similar to an ordered chain of phonological rewrite rules. Similarly, one can often define a lexicon in terms of regular languages of stems and affixes which are concatenated in the proper order (Beesley and Karttunen 2003:375).

Another useful property of finite-state machines is their capacity for optimization. Well-documented, mathematically proven methods exist to maximize the space and time efficiency of finite-state machines; as a result, FSAs and FSTs tend to be smaller and faster than alternative morphological processing algorithms (Roche and Schabes 1997:6-7; Beesley 2004c:3). FSAs can be made *deterministic* such that for any state, there is at most one transition out of that state with a particular label, and *minimal* such that no two states in the FSA are equivalent (Roche and Schabes 1997:6-7; Aho and Ullman 1995:547-555). FSTs can be minimized (Roche and Schabes 1997:53) and, frequently, made *sequential*²⁵ (deterministic for either the input or output language; Mohri 1997; Roche and Schabes 1997:43-46). Unambiguous transducers which are not sequentiable can be converted into *bimachines*, a pair of sequential transducers, one of which processes input from left to right, the other of which processes input from right to left; these are applied in sequence (Beesley and Karttunen 2003:77-78). The effect of all these optimizations is to create the smallest automaton or transducer such that for any input string there is at most one path of states and transitions through the automaton or transducer (in the case of a bimachine, one path through each transducer). Such a state machine can be applied in linear time²⁶ with little computational overhead.

Finally, unlike other approaches to computational morphology, FSTs are inherently bidirectional. Applied in one direction, they compute surface strings (generation); in the other, they compute underlying representations (analysis) (Beesley 2004c:3; Beesley and Karttunen 2003:14).

²⁵The terminology surrounding (partial) determinization of FSTs is particularly confusing. My use of the term “sequential transducer” follows Mohri (1997) and (Beesley and Karttunen 2003:76); Roche and Schabes (1997:43-46) use the term “subsequential transducer” to denote the same thing (and define a sequential transducer as something else; Mohri has his own definition of a subsequential transducer which is incompatible with that of Roche and Schabes). Roche and Schabes also note that what I am calling a sequential transducer is sometimes called a deterministic transducer.

²⁶An algorithm’s time efficiency is linear if the time required to obtain a result is directly proportional to the length of the input.

2.2.3.3 Finite-State Approaches to Morphophonology

Of the finite-state morphophonological models that have been proposed over the years, two have dominated all others: “two-level” systems based on Koskenniemi (1983), and so-called “lexical transducers” based on the work of Kaplan and Kay (published in 1994 but disseminated through conference presentations and personal communications as early as the 1980s) together with that of Karttunen et al. (1992). As mentioned previously, in his 1970 doctoral dissertation (published in 1972), C. Douglas Johnson showed that phonological rewrite rules from Generative Phonology are finite-state in power, and that in theory a set of ordered phonological rules could be composed into a single transducer. Kaplan and Kay rediscovered the same fact independently around 1980 (Karttunen and Beesley 2005:72). However, neither Johnson nor Kaplan and Kay were immediately successful in implementing a concrete finite-state computational morphology framework; most computers at the time lacked the speed and memory necessary to compile full rule systems together (Roark and Sproat 2007:111; Karttunen and Beesley 2005:72; Koskenniemi 1983:14). There was also an unresolved problem of how to prevent generative rules applied “in reverse” from generating multiple analyses for a given word, many of which would be spurious (Karttunen and Beesley 2005:72-74). Karttunen and Beesley give the example of two ordered rewrite rules (slightly simplified here): $n \rightarrow m / _ p$ and $p \rightarrow m / m _$. From the lexical form *kanpat*, these rules generate just one surface form, *kammat*. Applied in reverse, however, the rules do not specify that an ‘m’ must derive from either an ‘n’ or a ‘p’; thus, from the form *kammat*, the rules propose the forms *kanpat*, *kampat*, and *kammat*, the last two of which may be not be real lexical items (Karttunen and Beesley 2005:73).

In light of the computational limitations at the time, Koskenniemi (1983) devised a radically different model for finite-state morphophonology. The uniqueness of this model is perhaps best understood in contrast to generative phonology, the prevailing phonological theory at the time. In generative phonology, an underlying representation is gradually transformed into a surface form via an ordered series of rewrite rules, implying that underlying and surface forms are separated by several levels of intermediate representations. In Koskenniemi’s model, on the other hand, there are only two levels, lexical and surface, and Koskenniemi’s rules establish constraints on which lexical symbols may correspond to which surface symbols and vice versa. All symbol correspondences are one-to-one, but in order to facilitate insertions and deletions, Koskenniemi allows an empty symbol in one level to correspond to a non-empty symbol in the other level. Where generative rules apply in series, two-level rules are unordered and, in theory, apply in parallel. Parallel rule application, as implemented in Koskenniemi’s system, is theoretically equivalent to computing the intersection of each rule, which illustrates an important mathematical property of the two-level model: regular relations are not generally closed under intersection, but two-level rules are limited to same-length regular relations, which *are* closed under intersection (proofs can be found in Kaplan and Kay [1994:342-344] and Roark and Sproat [2007:106-108]).

Output in Koskenniemi’s system is further constrained by a lexicon. Any output candidate whose lexical level does not correspond to an entry in the lexicon is rejected (Karttunen and Beesley 2005:76); this resolves the problem illustrated in the *kanpat* example above. Lexical lookup occurs simultaneously with rule application, and is theoretically equivalent to composing the lexicon and the intersection of the rules (Karttunen and Beesley 2005:76). Because all interactions among rules and between the rules and the lexicon happen at runtime, it is not necessary to actually compile the system into a single FST (Roark and

Sproat 2007:111). Thus, Koskenniemi's model was able to overcome the limitations of 1980s computer hardware, although by doing so he lost much of the time efficiency normally associated with finite-state machines.

Two-level rules can be categorized into four types (Koskenniemi 1983:36-38; Karttunen and Beesley 1992, section 3.3; Roark and Sproat 2007:109), presented in Table 2.7. The notation $a:b$ represents a lexical symbol a and a surface symbol b whose relationship is to be constrained in some way by the rule; LC represents a left context, and RC a right context (both contexts can be specified in terms of zero or more lexical or surface symbols); and $_$ marks the position of $a:b$, defining the boundaries of LC and RC.

TABLE 2.7: Two-level rule types

Rule type	Notation	Meaning
Exclusion	$a:b \neq LC _ RC$	a cannot correspond to b in the given context
Context restriction	$a:b \Rightarrow LC _ RC$	only in the given context can a correspond to b (but a need not correspond to b in this context)
Surface coercion	$a:b \Leftarrow LC _ RC$	in the given context, a can only correspond to b (but this correspondence may also occur in other contexts)
Composite	$a:b \Leftrightarrow LC _ RC$	a always corresponds to b in the given context, and this correspondence does not occur in any other context

Linguists are not generally accustomed to thinking in terms of such rules, however, and the formalism can become particularly awkward when one needs to describe a correspondence involving more than one symbol (such as the rule $y \rightarrow ie / _ +s$).²⁷ Ruessink (1989) has proposed an alternative rule framework (for which Grimley-Evans et al. (1996) have written a compiler) to overcome some of these shortcomings.

Karttunen et al. (1992) point out two other considerable drawbacks to the two-level model (or at least common implementations of it): the arbitrary nature of lexical representations, and the treatment of morphological category tags as metadata rather than as part of the representation of the lexical form of words. An examination of the two-level lexical model described in Koskenniemi (1983:107-113) and a look at some sample two-level output will help demonstrate the import of these criticisms.

Koskenniemi's lexicon is implemented as a forest of tries (Karttunen and Beesley 2005:76). A trie is a tree which can represent a set of strings. Except for the root, each node in the tree is associated with a symbol. Any node may be designated as a leaf node; the path from the root to a leaf represents a string in the set. If two strings in the set share an overlapping prefix (such as *corn-* in *corner* and *cornea*), their representation in the trie will overlap for the nodes corresponding to that prefix. More information on tries can be found in a textbook on data structures (see for example Aho and Ullman 1995:223-234). In Koskenniemi's lexicon, each trie encodes a set of stems or suffixes, and the leaf node of each stem or suffix

²⁷This example comes from page 13 of the Stuttgart Finite State Transducer Tools (SFST) tutorial by Helmut Schmid: <ftp://ftp.ims.uni-stuttgart.de/pub/corpora/SFST/SFST-Tutorial.pdf>

may point to one or more “continuation classes”—tries containing suffixes that can be legally attached to the stem or suffix in question. The leaf node of each morpheme may also contain metadata about that morpheme, such as a canonical citation form (in the case of a lexeme) or a morphological category tag (in the case of a grammatical affix); this metadata is emitted as words are analyzed, making the analysis more meaningful. A small part of the nominal portion of Koskenniemi’s lexicon is presented in Figure 2.9. Trie names are in bold (I have taken the liberty of labeling the first trie “nouns”; other labels are Koskenniemi’s); dotted lines represent “branches” to continuation classes; boxed material is morphological metadata, and the nodes to which boxes are attached are leaf nodes. The abbreviations used in the boxes are as follows: S: substantive; GEN: genitive case; TRA: translative case; NOM: nominative case; ESS: essive case; PTV: partitive case; SG: singular.

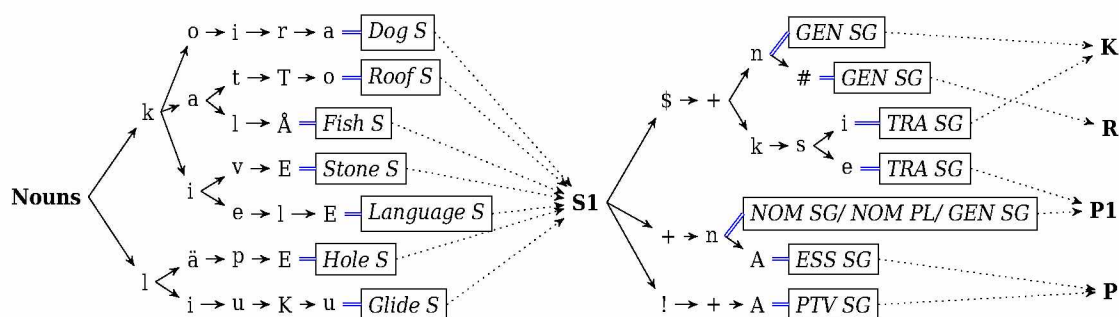


FIGURE 2.9: A portion of the forest-of-tries Finnish lexicon described in Koskenniemi (1983)

Table 2.8 shows a sample two-level analysis alongside a more traditional morphological analysis of the same word. The word being analyzed is *hakatuimmassa* ‘in the most hit/chopped up/beaten,’ which is a superlative passive past participle in the inessive case (see Karlsson 1999:194–210). The two-level analysis comes from Koskenniemi (1983:158); the translation and traditional analysis are my own.

TABLE 2.8: Two-level output compared to traditional morphological analysis

Two-level output	Traditional morphological analysis
hakKa\$t*\$ZTU\$+imPA\$+ssA	<i>hakat-ttu-imp-a-ssa</i>
Hit V PCP2 PSS SUP INE SG	hit-PAST.PASSIVE.PART-SUP-INESSIVE.SG

One of the first things one notices in both Figure 2.9 and Table 2.8 is the abundance of capital letters and other unusual symbols. In the case of the sample output, some of these symbols have no obvious correspondence with letters in the surface word. Finnish has a well-known series of consonant lenition/assimilation processes collectively known as “consonant gradation” (see Karlsson 1999:28–38). The infinitive stem *hakat-* has actually undergone consonant gradation, reducing an underlying ‘kk’ to a non-geminated ‘k’.

The passive marker *-ttu-* (or more specifically, its allomorph *-tu-*, since the stem ends in ‘t’) triggers the same gradation; thus, the surface word has only a singleton ‘k’. The superlative marker *-impa-* causes gradation of the ‘tt’, reducing it to ‘t’. Finally, the inessive case marker causes the ‘p’ in the superlative marker to weaken/assimilate to ‘m’. The symbols ‘K’, ‘Z’, ‘T’, and ‘P’ in the lexical representation of the word are sensitive to consonant gradation rules. Symbol ‘\$’ is used to trigger consonant gradation (in other words, gradation rules include ‘\$’ in the lexical part of the right context, and suffixes which cause gradation have a ‘\$’ in their lexical representation). ‘+’ is a boundary marker usually used before case endings; ‘*’ marks the left boundary of a passive suffix. The remaining special symbols, ‘A’ and ‘U’, are sensitive to vowel harmony rules. This inelegant lexical representation is necessary because of the restriction that all rules involve symbol correspondences, and because there can only be two levels of representation, one of which must be the surface level.

Ignoring the special symbols for a moment, the morphemic gloss presented on the second line of the two-level output in Table 2.8 is completely accurate and conveys the same information as the corresponding line in the traditional analysis; in fact, it is slightly more specific because it notes that the stem is a verb (the somewhat cryptic abbreviations are not due to any limitation of the two-level model; they are simply the convention Koskeniemi chose for his Finnish analyzer). For purposes of morphological analysis, the two-level model is satisfactory. As long as the user is familiar with the lexical and glossing conventions of the transducer (the user quite possibly being a computer program rather than a human), he/she/it can make sense of the output, and because the input consists of words represented in conventional orthography, producing “correct” input is not an issue.

Generation of surface forms is a different story. There are two possible forms that could be used as input, and each presents challenges. In order to supply an appropriate lexical representation (such as the form *hakKa\$t*\$ZTU\$+imPA\$+ssA* from Table 2.8 on the preceding page) one must be intimately familiar with the lexicon, particularly in cases where a morpheme has multiple allomorphs or non-alphabetic characters (the genitive singular endings in Figure 2.9 on the preceding page has both several allomorphs and non-alphabetic characters). A more natural way to specify input would be to supply a morphemic gloss, such as *Hit V PCP2 PSS SUP INE SG* from Table 2.8; the two-level engine could then substitute the corresponding lexical form and generate the surface form. But because the elements of the gloss are encoded as metadata on leaf nodes rather than as part of the structure of the lexicon, there is no easy way to look them up (Karttunen et al. 1992:142).²⁸

To remedy the problems of arbitrary lexical representations and hard-to-look-up morphological category tags, Karttunen et al. (1992) propose that all inflected surface forms of a word be mapped onto their respective lemma, and that for inflectional morphemes, grammatical tags be used instead of abstract underlying forms (this would make the grammatical tags an integral part of the lexicon rather than associated metadata). The word analyzed in Table 2.8 would then have a lexical representation along the lines of *hakata+part+past+pass+sup+iness+sg* (*hakata* is the infinitive form of the verb and the form conventionally used in Finnish dictionaries). Within the framework of two-level morphology, these proposals present a challenge: the more dissimilar surface and lexical representations of a word are, the more difficult it becomes to write adequate symbol correspondence rules between the two forms (Karttunen et al.

²⁸The task is somewhat analogous to searching for the French translation of an English word using a French to English dictionary: chances are that many entries will have to be examined before the relevant one is found.

1992:142). The solution offered by Karttunen et al. is to abandon the two-level limit in favor of serial rule composition.

The most important contribution of Karttunen et al. (1992) is to extend the compositional model of Johnson and Kaplan and Kay to the lexicon. Composing the lexicon with the morphophonological rules effectively computes the surface and lexical representations of all legal forms of all words described by the lexicon. The resulting “lexical transducer” (Karttunen 1994) is considerably more efficient than two-level morphological engines.²⁹ In the original two-level model, rules are *theoretically* intersected into a single transducer, and the lexicon is *theoretically* composed to the lexical side of this transducer, but in actual practice, each rule is implemented as a separate transducer, the lexicon is distinct from the rules, and each must be traversed separately. In a lexical transducer, the lexicon and all rules are combined into a single transducer, not just theoretically but in reality.

A surprising side-effect of composing the lexicon with the phonological rules is that (aside from trivial cases) the resulting transducer is smaller than the sum of the sizes of the lexicon and the phonology as individual state machines. In fact, the lexical transducer is often smaller than the size of the phonological transducer alone, even in cases where the lexicon is quite large (see Karttunen et al. 1992:146; Beesley and Karttunen 2003:xvi-xvii). In order to understand how this is possible, recall from the *kanpat/kammat* discussion on page 45 that in the two-level model, the lexicon serves to rule out spurious analyses generated by decontextualized rules. The lexicon serves the same purpose in a lexical transducer. However, nonexistent candidates are not simply discounted as the transducer is applied; they are actually eliminated from the transducer through the process of composition.

Lexical transducers thus represent a more time- and space-efficient alternative to two-level morphology engines, allowing more natural lexical entries and a more flexible way to write and combine phonological rules. In fairness to Koskenniemi, it must be noted that the computational resources required to compile lexical transducers were not available at the time (Roark and Sproat 2007:111), and that two-level phonology represents an ingenious workaround for these computational limitations. But in most cases today, the advantages of the compositional model outweigh those of the two-level model. This is even more true thanks to the introduction of more powerful rule types such as conditional and non-conditional string replacement (Karttunen 1997), which are neither restricted to single symbols nor same length relations, and which (unlike two-level rules) closely resemble the generative rules familiar to most linguists.

Although the compositional approach provides several advantages over the two-level model, it should be seen as subsuming rather than superseding it. Some morphophonological phenomena are more easily modeled using two-level rules than rewrite rules (Beesley and Karttunen 2003:384–385). Since two-level rules can be compiled into finite-state transducers, they can be combined with other transducers using any appropriate operation under which transducers are closed (Roark and Sproat [2007:113]; this should also be obvious from Karttunen et al. [1992]).

Several toolkits for developing lexical transducers exist. Foremost among these is probably the Xerox Finite State Tools (XFST), described in tutorial fashion in Beesley and Karttunen (2003) and much more briefly in (Beesley 2004c:3–4).³⁰ Similar toolkits include the AT&T FSM library (<http://www2.research>.

²⁹Alegria et al. (1996:201) report that their lexical transducer for Basque performed 500 times faster than the equivalent two-level engine.

³⁰Much of the research on modern finite-state morphology has been carried out at Xerox and its subsidiaries and spinoffs; current

att.com/~fsmtools/fsm/), the Finite State Automata Utilities (Fsa Utils 6) by Gertjan van Noord and Dale Gerdemann (<http://www.let.rug.nl/vannoord/Fsa/>), and the Stuttgart Finite State Transducer Tools (SFST) by Helmut Schmid (<http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html>), among others. These toolkits allow linguists to create morphological engines in high-level terms of set-theoretic operations on regular languages and relations rather than in low-level terms of states and transitions. Kaplan and Kay (1994:376) make a forceful case for the necessity of such toolkits:

The common data structures that our programs manipulate are clearly states, transitions, labels, and label pairs—the building blocks of finite automata and transducers. But many of our initial mistakes and failures arose from attempting also to think in terms of these objects. The automata required to implement even the simplest examples are large and involve considerable subtlety for their construction. To view them from the perspective of states and transitions is much like predicting weather patterns by studying the movements of atoms and molecules or inverting a matrix with a Turing machine. The only hope of success in this domain lies in developing an appropriate set of high-level algebraic operators for reasoning about languages and relations and for justifying a corresponding set of operators and automata for computation.

2.3 The Xerox Finite State Tools

The Xerox Finite State Toolkit (XFST) is probably the most widely used software for creating lexical transducers (Kornai 1999:4). It provides two languages, *xfst*³¹ and *lexc*, designed to be used in tandem.³² *Lexc* is a right-recursive phrase structure grammar used to create lexicons (Beesley and Karttunen 2003:203), and *xfst* is a regular expression language for writing morphographemic rules and combining these together (Beesley and Karttunen 2003:81). This section will provide an overview of how lexical transducers are constructed using *xfst* and *lexc*, with a special emphasis on features and techniques that will be useful in modeling Iñupiaq morphology.

2.3.1 Multicharacter Symbols

The fundamental unit of input (and output, if applicable) in a finite-state machine is the symbol. Sometimes it is convenient for several characters (or code points) to be treated as a single symbol:

- In the orthographic traditions of some languages, digraphs are considered distinct letters. For example, Croatian digraphs ⟨dž⟩, ⟨lj⟩, and ⟨nj⟩ are considered single letters for purposes of collation. Computational morphologists may prefer to treat them as single symbols in a transducer as well.
- Some accented letters are not given their own slot in the Unicode standard, and can only be represented as a sequence of code points; for example, the letter ⟨x̂⟩, used in Aleut, must be composed

and former Xerox scientists working on finite-state morphology include Ronald Kaplan, Martin Kay, Lauri Karttunen, Kenneth Beesley, Annie Zaenen, and undoubtedly many others.

³¹I will use lower case letters to denote the *xfst* regular expression language, and capital letters to denote the XFST software. This convention is also used in Beesley and Karttunen (2003).

³²XFST also provides a third, less commonly used language called *twolc* for specifying two-level rules. *twolc* can be used in conjunction with *xfst* and/or *lexc*, but most linguists prefer not to deal with two-level rules.

from code points U+0078 (lower-case x) and U+0302 (combining circumflex).³³ If a transducer is to contain an accented letter such as this, it makes sense to declare the sequence a multicharacter symbol (but see remarks on {‡} in Section 2.4.3.1, pages 61–62).

- By Xerox convention, grammatical tags such as +3sg are treated as one symbol.
- “Rule triggers” (to be discussed in Section 2.3.4 on pages 54–55), which are attached to specific morphemes to mark them as subject to particular morphologically or lexically conditioned alternations, are most conveniently treated as single symbols even if they consist of several characters.
- Flag diacritics (to be discussed in Section 2.3.5 on pages 55–58) must be treated as single symbols in order to function properly.

Mechanisms exist in both xfst and lexc for designating strings of characters as multicharacter symbols. In xfst, symbols are normally separated by whitespace or xfst syntax characters, so any time a string of several non-whitespace, non-syntactic characters is encountered, it is treated as a multicharacter symbol. In lexc, symbols are not separated by whitespace; by default, each character is treated as a distinct symbol. Multicharacter symbols in a lexc file must be explicitly declared at the beginning of the file; this allows lexc to properly tokenize lexc entries into the intended symbols.

2.3.2 Lexicon Creation

In lexc, lexicons are specified in terms of classes of formatives.³⁴ A formative class is a named set of string/continuation class pairs. Typically, the strings are word formatives, but they may also include special symbols such as grammatical tags, rule triggers (see Section 2.3.4 on pages 54–55), or flag diacritics (see Section 2.3.5 on pages 55–58), or they may be empty strings. A continuation class is either an end-of-word indicator or the name of a formative class. One formative class is designated as root. A word in the lexicon consists of a string from the root class, followed by a string from the continuation class of the first string, followed by a string from the continuation class of that string, and so on until the final string continues to the end of the word rather than to another formative class.

Figure 2.10 on the following page illustrates the concept of formative classes and continuations as applied to the French infinitive verbs *couper*, *couvrir*, *construire*, *voir*, *lever*, *recouper*, *recouvrir*, *reconstruire*, *revoir*, *relever*, *découper*, *découvrir*, and *déconstruire*. The base forms in this example (*couper*, *couvrir*, *construire*, *voir*, and *lever*) are all verbs in their own right; each of them may also be prefixed with *re-*. But *voir* and *lever* may not be prefixed with *dé-*, so we must distinguish two classes of verbs. Since the root class must begin at the left edge of the word, it contains the two prefixes as well as an empty string (represented by the symbol ‘ε’) to accommodate unprefixed verbs. The prefix *dé-* continues directly to the set of verbs it may prefix (Verbs1), but the prefix *re-* and the empty prefix continue to an intermediate class (AllVerbs) which contains only empty strings, each of which continues to a different class of verbs. All stems are marked as end-of-word strings (#).

In addition to regular languages, lexc allows one to define regular relations. This facility is not intended for creating complete lexical transducers; xfst provides a more practical interface for describing most

³³See the Unicode Consortium’s “Frequently Asked Questions: Characters and Combining Marks,” http://unicode.org/faq/char_combmark.html.

³⁴Beesley and Karttunen (2003) call these “lexicons,” which leads to ambiguity: one defines lexicons (in the more traditional sense) in terms of lexicons (in the innovative sense). For this reason, I prefer to call these formative classes.

Root	AllVerbs	Verbs1	Verbs2
re <i>AllVerbs</i>	ε <i>Verbs1</i>	couper #	voir #
dé <i>Verbs1</i>	ε <i>Verbs2</i>	couvrir #	lever #
ε <i>AllVerbs</i>		construire #	

FIGURE 2.10: Formative classes defining a regular language containing some French infinitive verbs

alternations. Relations in *lexc* are best used to account for idiosyncratic differences between underlying and surface forms, in such cases as suppletion, irregular members of paradigms, or the mapping of abstract tags such as *+Inf* onto surface strings such as *er* (Beesley and Karttunen 2003:203, 226–227, 230–232). It is possible to define quite complex relations in a *lexc* entry, but for this thesis we limit ourselves to the most basic kind, where one simply specifies an upper string and a lower string which correspond to each other.

Figure 2.11 illustrates formative classes that define a regular relation. This example builds upon the previous one, and classes *Root* and *AllVerbs* remain unchanged from that example and have not been reproduced here. The relation defined in this example contains the same verbs as the previous example, but in addition to infinitives, this relation contains past participles (inflected for number and gender) and present indicative forms for 1st person singular and plural subjects. The upper language consists of verbs in their infinitive forms followed by grammatical tags³⁵ (e.g., *revoir+PstPart+F+Pl*, *construire+Ind+Prs+1sg*). The lower language consists of the corresponding surface forms (e.g., *revues*, *construis*). In the figure, colons separate corresponding upper and lower strings, with upper strings on the left.

Verbs1	Verbs2	erInflection
couper:coup <i>erInflection</i>	voir:v <i>voirInflection</i>	+Inf:er #
couvrir:couv <i>vrirInflection</i>	lever:lev <i>erInflection</i>	+PstPart:é <i>Gender</i>
construire:construi <i>uireInflection</i>		+Ind+Prs+1sg:e #
		+Ind+Prs+1pl:ons #

vrirInflection	uireInflection	voirInflection
+Inf:rir #	+Inf:re #	+Inf:oir #
+PstPart:ert <i>Gender</i>	+PstPart:t <i>Gender</i>	+PstPart:u <i>Gender</i>
+Ind+Prs+1sg:re #	+Ind+Prs+1sg:s #	+Ind+Prs+1sg:oie #
+Ind+Prs+1pl:rons #	+Ind+Prs+1pl:sons #	+Ind+Prs+1pl:oyons #

Gender	Number
+M:ε <i>Number</i>	+Sg:ε #
+F:e <i>Number</i>	+Pl:s #

FIGURE 2.11: Modified formative classes defining a regular relation between grammatical analysis and surface forms of some French verbs

³⁵Tags used in this example are *+Inf* infinitive, *+PstPart* past participle, *+Prs* present, *+Ind* indicative, *+1sg* 1st person singular, *+1pl* 1st person plural, *+M* masculine, *+F* feminine, *+Sg* singular, *+Pl* plural.

This should not be taken as the only or best way to organize a French lexicon, but it will serve to illustrate some important points. The relation mechanism serves two purposes in this example: mapping citation forms (e.g., *couper*, *voir*) onto more stem-like forms (e.g., *coup*, *v*), and mapping grammatical tags (e.g. *+Inf*, *+Pl*) onto concrete strings. Tags *+M* and *+Sg* correspond to empty surface strings, as French lacks overt masculine and singular markers for adjectives.

Verbs *couper* and *lever* have very similar conjugation patterns, and both continue to the *er*Inflection class. However, the resulting forms are not entirely correct for *lever*; the formative classes in the example will define the surface form **leve* for the 1st person singular indicative present form, instead of the correct form *lève*. This situation could be remedied by creating a class *everInflection*, but in fact the alternation between [ə] (as in *lever*) and [ɛ] (as in *lève*) applies to all verbs ending in *-er* which have a schwa as the nucleus of their penultimate syllable (see Bescherelle 1995:24, 26, 27). Rather than create separate classes for *-ecer*, *-eler*, *-emer*, *-ener*, *-eper*, *-erer* *-eser*, *-eter*, *-ever*, and *-evrer*, it would be much simpler to use the *erInflection* class for all of these cases, and compose the resulting lexicon with a rule written in *xfst* to change the vowel when appropriate.³⁶

2.3.3 Morphographic Rules

Morphographic rules are expressed in the *xfst* language and can be divided into two types: those which define languages, and those which define relations. Rules which define languages are conceptually similar to regular expressions found in many modern programming languages, except that they may be combined with other rules using a number of operations not commonly found in programming languages (such as composition and intersection). They may include strings of symbols, wildcards, iterators (indicating, for example, zero or one, zero or more, or one or more instances of a particular sequence of symbols), and operators for operations under which regular languages are closed (see pages 43–44). Rules which define relations most commonly use a mechanism called the replace operator (Karttunen 1997; Beesley and Karttunen 2003:66–74, 132–137), which defines directional correspondences between an upper and a lower language. The replacement may be obligatory or optional, and it may be conditional or unconditional. It is defined in terms of the universal language (the set of all possible strings composed of symbols from some finite alphabet), so that any string not specifically targeted by a given rule automatically corresponds to an identical string.

An unconditional rule has three parts: a direction (leftward or rightward) and two expressions, which I will call *UpperExp* and *LowerExp*, each defining a regular language. In *xfst* notation, an unconditional rule takes the form *UpperExp Direction LowerExp*; *Direction* is either *->* or *<-*, and is surrounded by parentheses if the rule is optional. A obligatory unconditional rightward rule specifies that every string in the upper language corresponds to an identical string in the lower language, unless it contains one or more substrings defined by *UpperExp*; in that case, it corresponds to all lower-language strings where each substring from *UpperExp* has been replaced by a string defined by *LowerExp*. An optional rightward rule additionally specifies that every string in the upper language containing substrings defined by *UpperExp* also corresponds to an identical string in the lower language. A obligatory unconditional leftward rule is analogous to a rightward rule, but substrings from *LowerExp* in lower-language strings are replaced in

³⁶The complete solution to this problem is actually slightly more complex than this, as will be explained in Section 2.3.4 below.

the corresponding upper-language strings with strings defined by UpperExp.

To illustrate, consider a obligatory unconditional rule whose direction is rightward, where UpperExp defines the set {a, e, i, o, u}, and where LowerExp defines the set {V}. In xfst notation, this rule would be written $[[a | e | i | o | u] \rightarrow V]$ (square brackets group expressions; the pipe symbol (|) is the union operator). The upper language of this rule would be the universal language; the lower language would be the set of all strings which do not contain any of the symbols *a*, *e*, *i*, *o*, or *u*. Some string correspondences defined by this relation are (*bcdfg*, *bcdfg*), (*yes*, *yVs*), and (*congratulations*, *cVngrVtVlVtVVns*). Note that obligatory rules may restrict the language where replacement takes place, but will not restrict the other language. In the rule defined in our example, the string *yes* in the upper language corresponds only to the string *yVs* in the lower language, but the lower language string *yVs* corresponds to several upper language strings: *yes*, *yas*, *yis*, *yos*, *yus*, and its identity, *yVs*. The presence of the identity may seem counterintuitive, but it is a consequence of directed (as opposed to bidirectional) replacement (see the *kanpat/kammatt* discussion on page 45).

Conditional rules have the same three parts as unconditional rules, but they also specify a left and right context, which I will call LeftExp and RightExp; they also specify in which language (upper or lower) each context should apply; in this thesis, all conditional rules will specify upper-language contexts. Either context may be unspecified (if both are unspecified, the rule is equivalent to an unconditional rule). The xfst convention for conditional rules with upper-language contexts is *UpperExp Direction LowerExp || LeftExp _ RightExp*. A rightward conditional rule where both contexts refer to the upper language specifies that every string in the upper language corresponds to an identical string in the lower language, unless it contains one or more substrings defined by UpperExp which are immediately preceded by a substring defined by LeftExp and immediately followed by a substring defined by RightExp; in this case, it corresponds to all lower-language strings where each substring from UpperExp occurring in the defined context is replaced by a substring defined by LowerExp. Note that the contexts are not replaced; only the substring defined by UpperExp is replaced.

The rule $n \rightarrow \tilde{n} / i _$, shown as a finite-state transducer in Figure 2.7 on page 41, can be specified as a conditional replace rule. UpperExp is {n}, LowerExp is {ñ}, LeftExp is {î}, and RightExp is unspecified. The direction is rightward and the rule is obligatory; the contexts refer to the upper language. This rule would be written $[n \rightarrow \tilde{n} || \hat{i} _]$. The rule establishes correspondences between all strings containing the sequence *îñ* in the upper language and strings in the lower language that are identical except that *îñ* has been replaced with *îñ*. The sequence *îñ* in lower-language strings corresponds to both *îñ* and *îñ* in upper-language strings.

Rules which define relations may be combined using operations under which regular relations are closed (see pages 43–44).

2.3.4 Rule Triggers

Some morphologically and lexically conditioned alternations are best handled through the continuation class mechanism, but others are easier to model through conditional xfst rules. For such rules to apply properly, the affected words or morphemes must be distinguished from all others. This can be done using symbols which are not part of the orthography of the language. These symbols are included in the

specification of the affected words (this will be illustrated below). Rules can then make reference to these symbols, either as part of the context (LeftExp or RightExp) or in the expression defining substrings to be replaced (UpperExp or LowerExp). Uí Dhonnchadha (2003:46) refers to such symbols as “rule triggers”; Beesley (2003:22) and Alegria et al. (1996:195,202) also report using symbols of this type in their transducers, and Langgård and Trosterud (n.d.; see Section 2.4 on pages 60–66) use them in their Iñupiaq transducer, where they refer to them as “dummies”. An illustration of rule triggers is given below.

In the discussion of French verbs in Section 2.3.2, it was mentioned that all French verbs ending in *-er* which have a schwa as the nucleus of the penultimate syllable are subject to an alternation where the schwa becomes [ɛ], and that this alternation was more easily handled with a general rule than through several very similar formative classes. However, as is frequently the case with French, the orthographic reflex of this phenomenon is slightly more complex than the phonetic phenomenon itself. For most of the verbs affected by this alternation, the change in vowel quality is represented by changing the ⟨e⟩ of the infinitive form to ⟨è⟩: *lever/lève*. Verbs ending in *-eler* and *-eter* generally double the consonant following the alternating vowel, leaving the ⟨e⟩ untouched: *appeler/appelle*. But a handful of verbs ending in *-eler* and *-eter* mark the change with ⟨è⟩ and leave the following consonant single: *acheter/achète*.

To properly model this fact, we need two rules, one to map ⟨e⟩ to ⟨è⟩, and the other to map the consonant following the ⟨e⟩ to its doubled equivalent. For most of the affected verbs, the first rule can be applied based on phonological criteria alone, but for verbs ending in *-eler* and *-eter*, it is impossible to know *a priori* which rule should apply. We can address this situation by marking the consonant-doubling verbs with rule triggers (e.g., *appelDOUBLECer*, *étinceDOUBLECer*, *marquetDOUBLECer*, where *DOUBLEC* has been declared a multicharacter symbol) and writing the consonant-doubling rule so that this symbol appears in the right context of the rule. (We will also want an unconditional rule to delete the *DOUBLEC* symbol; this rule should probably be composed immediately below the consonant doubling rule.) The vowel-changing rule does not need to refer to the *DOUBLEC* symbol in any way; it can be written in such a way that ⟨e⟩ is mapped to ⟨e⟩ (the default mapping) rather than ⟨è⟩ if a double consonant follows. As long as the rules are ordered so that the consonant-doubling rule applies first, each rule will apply only to verbs to which it pertains.

2.3.5 Flag Diacritics and Filters

Long-distance morphological dependencies are not easily encoded using continuation classes (Beesley and Karttunen 2003:247–248). For example, in Amharic, definite direct objects are marked with a suffix ⟨-n⟩, but indirect objects are marked with a prefix ⟨lā-⟩.³⁷ For purposes of illustration, I will assume that a noun may bear neither or one marker, but not both. Such a constraint can theoretically be modeled using continuation classes, as shown in Figure 2.12 on the following page (see also Beesley and Karttunen 2003:250), but the result is redundant and inelegant, as whole formative classes must be duplicated.³⁸ The formative classes in Figure 2.13 on the next page resolve the duplication, but fail to account for the discontinuous dependency and therefore overgenerate, producing illegal forms such as **lā-wānd-u-n* ‘IND.OBJ-boy-the-DIR.OBJ’ (see Beesley and Karttunen 2003:251).

³⁷ Amharic examples follow the transliteration conventions of Leslau (1995).

³⁸ A complete model of Amharic nouns would contain even more duplication, as one adds suffixes for grammatical number and possessor marking, both of which intervene between the noun stem and the direct object marker.

Root <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+IndObj:lä</td><td style="padding: 2px;"><i>LäNouns</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;"><i>Nouns</i></td></tr> </table>	+IndObj:lä	<i>LäNouns</i>	ε	<i>Nouns</i>	LäNouns <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">wänd</td><td style="padding: 2px;"><i>LäArt</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">wättaddär</td><td style="padding: 2px;"><i>LäArt</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ğəb</td><td style="padding: 2px;"><i>LäArt</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">mäšhaf</td><td style="padding: 2px;"><i>LäArt</i></td></tr> </table>	wänd	<i>LäArt</i>	wättaddär	<i>LäArt</i>	ğəb	<i>LäArt</i>	mäšhaf	<i>LäArt</i>	Nouns <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">wänd</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">wättaddär</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ğəb</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">mäšhaf</td><td style="padding: 2px;"><i>Art</i></td></tr> </table>	wänd	<i>Art</i>	wättaddär	<i>Art</i>	ğəb	<i>Art</i>	mäšhaf	<i>Art</i>
+IndObj:lä	<i>LäNouns</i>																					
ε	<i>Nouns</i>																					
wänd	<i>LäArt</i>																					
wättaddär	<i>LäArt</i>																					
ğəb	<i>LäArt</i>																					
mäšhaf	<i>LäArt</i>																					
wänd	<i>Art</i>																					
wättaddär	<i>Art</i>																					
ğəb	<i>Art</i>																					
mäšhaf	<i>Art</i>																					
LäArt <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+Art:u</td><td style="padding: 2px;">#</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;">#</td></tr> </table>	+Art:u	#	ε	#	Art <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+Art:u</td><td style="padding: 2px;"><i>DirObj</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;">#</td></tr> </table>	+Art:u	<i>DirObj</i>	ε	#	DirObj <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+DirObj:n</td><td style="padding: 2px;">#</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;">#</td></tr> </table>	+DirObj:n	#	ε	#								
+Art:u	#																					
ε	#																					
+Art:u	<i>DirObj</i>																					
ε	#																					
+DirObj:n	#																					
ε	#																					

FIGURE 2.12: Accounting for a discontinuous dependency using formative classes and continuations

Root <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+IndObj:lä</td><td style="padding: 2px;"><i>Nouns</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;"><i>Nouns</i></td></tr> </table>	+IndObj:lä	<i>Nouns</i>	ε	<i>Nouns</i>	Nouns <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">wänd</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">wättaddär</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ğəb</td><td style="padding: 2px;"><i>Art</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">mäšhaf</td><td style="padding: 2px;"><i>Art</i></td></tr> </table>	wänd	<i>Art</i>	wättaddär	<i>Art</i>	ğəb	<i>Art</i>	mäšhaf	<i>Art</i>	Art <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+Art:u</td><td style="padding: 2px;"><i>DirObj</i></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;">#</td></tr> </table>	+Art:u	<i>DirObj</i>	ε	#	DirObj <table style="width: 100%; border-collapse: collapse;"> <tr><td style="border-right: 1px solid black; padding: 2px;">+DirObj:n</td><td style="padding: 2px;">#</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px;">ε</td><td style="padding: 2px;">#</td></tr> </table>	+DirObj:n	#	ε	#
+IndObj:lä	<i>Nouns</i>																						
ε	<i>Nouns</i>																						
wänd	<i>Art</i>																						
wättaddär	<i>Art</i>																						
ğəb	<i>Art</i>																						
mäšhaf	<i>Art</i>																						
+Art:u	<i>DirObj</i>																						
ε	#																						
+DirObj:n	#																						
ε	#																						

FIGURE 2.13: Formative classes which fail to account for discontinuous dependencies

If not for the problem of overgeneration, it would be preferable to create lexicons as in Figure 2.13 rather than as in Figure 2.12. *xfst* makes this possible by providing two equivalent mechanisms to control this type of overgeneration. The first, which Beesley and Karttunen (2003:249–254) call a filter, involves creating a rule to define a regular language which contains all strings except the offending ones (e.g. **läwändun*) and composing this rule with the lexicon; the composition process eliminates the ungrammatical words from the transducer. In the case of the Amharic example, the rule would define the complement of the language in which all strings contain the multicharacter symbol *+IndObj* followed by zero or more symbols followed by the multicharacter symbol *+DirObj* (see Beesley and Karttunen 2003:252–253; Beesley 1998:122–123).

Filters provide an elegant, finite-state solution to the problem at hand, but sometimes they lead to a dramatic increase in the size of a transducer (Beesley 1998:122; Beesley and Karttunen 2003:339), which may make compilation of the transducer impractical or, depending on hardware constraints, impossible. For this reason, *xfst* includes an alternative method of disallowing illegal combinations of discontinuous morphemes: flag diacritics. Flag diacritics are special multicharacter symbols which set, unset, or test memory registers. They are incorporated into the appropriate entries in the lexicon, similarly to rule triggers (see section 2.3.4). Unlike other multicharacter symbols, flag diacritics are usually invisible to the end-user, but within the transducer they label transitions between states, just as any other symbol. As *xfst* encounters flag diacritics while traversing a particular path in an FST, it performs the memory actions they specify. If *xfst* encounters a flag diacritic testing a register for a certain value and the test fails, that path through the transducer is considered a failure and *xfst* backtracks in an attempt to find an alternative path.

xfst provides several types of flag diacritics, enumerated in Table 2.9 on the following page. In our

Amharic example, we could create a flag diacritic @PCASE.MARKED@ (that is, a Positive-type diacritic that would set the register CASE to the value MARKED) to attach to the indirect object marker, and another diacritic @DCASE.MARKED@ (to disallow a CASE value of MARKED) to attach to the direct object marker. The formative classes from Figure 2.13 on the preceding page could be revised as in Figure 2.14 to include these flag diacritics.

TABLE 2.9: Flag diacritic types available in xfst

Type	Arguments	Function
Positive	<i>Register, Value</i>	Sets or resets <i>Register</i> to <i>Value</i>
Negative	<i>Register, Value</i>	Sets or resets <i>Register</i> to the complement of <i>Value</i> (i.e., all values other than <i>Value</i>)
Clear	<i>Register</i>	Sets or resets <i>Register</i> to no particular value
Require	<i>Register, Value</i>	Tests whether <i>Register</i> is set to <i>Value</i>
Disallow	<i>Register, Value</i>	Tests whether <i>Register</i> is set to a value other than <i>Value</i>
Unify	<i>Register, Value</i>	If <i>Register</i> is not set to a specific value, sets it to <i>Value</i> ; otherwise, tests whether <i>Register</i> is set to <i>Value</i>

Root	Nouns	Art	DirObj
@U.CASE.MARKED@+IndObj:@U.CASE.MARKED@lä	Nouns	+Art:u	DirObj
ε	Nouns	ε	#
			@D.CASE.MARKED@+DirObj:@D.CASE.MARKED@n
			ε
			#

FIGURE 2.14: Formative classes which account for discontinuous dependencies using flag diacritics

In the transducer, the diacritics are treated as symbols, so the transducer will contain lower-language strings such as *māshafu@D.CASE.MARKED@n* and *@U.CASE.MARKED@läğəb*, corresponding to upper-language strings such as *māshaf+Art@D.CASE.MARKED@+DirObj* and *@U.CASE.MARKED@+IndObjğəb* (note that each diacritic must appear on both the upper and lower sides of the transducer, since the strings they are meant to suppress must be suppressed in both the upper and lower languages.) Strings representing ungrammatical words, such as *@U.CASE.MARKED@läwättaddäru@D.CASE.MARKED@n*, will also be present in the transducer, but will never be returned to the end user because the disallow test of the second flag diacritic will always fail.

One potential pitfall associated with flag diacritics is that, while they are generally hidden from the end user, their presence must be taken into account by any rules that will be composed with the transducer.³⁹

³⁹Similar statements could be made about rule triggers. An obvious difference between rule triggers and flag diacritics is that

For example, a rule which includes the substring *un* in its context will never fire for any of the words defined in Figure 2.14; the rule should specify the context *u@D.CASE.MARKED@n* instead. It is easy to forget to do this, and bugs that result from not taking flag diacritics into account are very hard to track down. Another drawback of flag diacritics is a loss of time efficiency, since they create the possibility that several disallowed paths will have to be evaluated in addition to all allowed paths (Beesley and Karttunen 2003:360).

While the flag diacritic mechanism is not finite-state in nature (it goes beyond the machinery of states and transitions), it is finite-state in power, and functionally equivalent to filters (Beesley and Karttunen 2003:359). Because of this equivalence, a transducer containing flag diacritics can be converted to a filtered, flag-free transducer. The conversion process is not necessary, and because it does away with the additional space efficiency flag diacritics provide, in many cases is inadvisable. But where space is not an issue, eliminating flags can be a useful strategy for morphologists who find it easier to constrain discontinuous dependencies with diacritics rather than filters, but who do not want to deal with diacritics in their rules.

2.3.6 Dictionary Downtranslation

As was made clear in Section 2.2.3.3 on pages 45–50, a lexicon is a key component in linguistically motivated computational morphology. But the need for structured lexical information is not limited to morphological parsers; many NLP applications require some kind of lexical database, although the specific requirements vary from application to application. A lexical transducer requires relatively simple lexical data—essentially a set of fully inflected words and associated lemmata and grammar tags. If one is using XFST, the most obvious way to structure the necessary lexical data is lexc formative classes. However, Beesley (2004a, c, 2003) advises against creating lexc files from scratch, calling them a “dead-end” (2004a:2) because the format is specific to XFST and the data are too sparse to be very useful to other applications. (To these criticisms I would add that lexical data is rarely presented in terms of formative classes of the type used by lexc, and thus creating a lexc file generally involves completely reorganizing one’s data.) As an alternative to developing directly in lexc, Beesley recommends creating lexical resources in XML, a meta-language for data markup which can be used to represent data sets of arbitrary complexity. A single XML database can contain all the data necessary for whichever NLP applications one wishes to develop. For example, if there are plans to develop a morphological parser, syntactic parser, and electronic dictionary for a language, then each entry in the XML database might include a lemma, grammatical category, inflectional pattern, valency information, definitions or glosses for each sense, semantic frame data (see for example Johnson and Fillmore 2000; Atkins 1996), and so on. Some data are not needed for some applications; for example, semantic frames are irrelevant to morphological parsing, and inflectional patterns have no bearing on syntactic parsing. But some data, like lemmata or grammatical categories, are important for several applications.

Before the data in an XML file can be used in a morphological parser, however, it needs to be “downtranslated” into a format XFST can understand (Beesley 2004a). The downtranslation process involves

the former are created for use in rules, whereas the latter are not. However, depending on the design of the transducer, it may still be necessary to write rules in which rule triggers appear in the context not because the phenomenon being modeled is sensitive to them but simply because they happen to occur in the character string between symbols to which the phenomenon is sensitive.

discarding XML markup as well as any data not pertinent to morphological parsing (such as glosses) and reformatting the pared-down data in terms of formative classes and continuations. There are a number of possible ways to accomplish this task; Beesley (2004a) recommends writing a computer program using a language with strong string-handling facilities (such as Perl or Python) and an XML processing library (such as Perl's XML::Twig⁴⁰ or Python's pulldom⁴¹).

2.3.7 Stem Guessers

While a lexicon is important for eliminating spurious parses, no lexicon can contain all the words in a language. To a certain degree, it is possible to compensate for this shortcoming using a technique called stem or root guessing. First developed by Black et al. (1991) for use in speech-synthesizing transducers, the technique has proven beneficial for other morphological parsers as well (see for example Alegria et al. 1996:200; Bosch and Pretorius 2003:33; Beesley 2003:24–25). It involves defining, for each open category of stems, a transducer that (ideally) generates all phonotactically possible members of that class (Beesley and Karttunen 2003:450). In the upper language of this phonotactic transducer, all strings terminate with a tag which labels them as guesses rather than as official entries in the lexicon; this tag corresponds to an empty string in the lower language (Beesley and Karttunen 2003:446). The phonotactic transducer is made a member of an appropriate class of stems, and designated to continue to the class containing all affixes which may be attached to that particular category of stems (see Beesley and Karttunen 2003:445). By filtering out all paths in the transducer which do not contain the guess tag, one obtains a transducer which produces only guesses (by the same token, one can filter out all paths containing the tag and obtain a transducer which produces no guesses; see Beesley and Karttunen [2003:446, 448]).

A guessing transducer can be used as a backup to a standard lexical transducer, made to apply only when the non-guessing version cannot propose an analysis (Beesley 2003:24–25; Beesley and Karttunen 2003:449–450). An obvious limitation of stem guessing based on native phonotactics is that stems borrowed from other languages may not be recognized; a possible workaround would be to define a second, more generous guesser (possibly limited to stems shorter than some given length) and configure the system so that the second guesser only applies in cases where the non-guessing transducer and the native-phonotactics guesser both fail to produce an analysis.

A well-designed stem guesser constrains the set of analyses by eliminating parses which are impossible phonotactically or inflectionally. However, for most input strings, a guesser is likely to generate multiple analyses, some of which will be better than others. Paradoxically, the number of incorrect analyses increases with the number of affixes included in the lexicon (see Black et al. 1991:105). For their morphological analyzer for speech synthesis of English, Black et al. (1991:104) propose a set of heuristics to rank analyses with guesses: length of unknown root, structural ordering rules (for example, analyses involving prefixes were given priority over analyses involving suffixes), and affix frequency. The analysis with the highest total ranking (calculated as a weighted sum of the individual scores from each heuristic) was selected as most probable. In a test of 200 words, this ranking procedure selected the correct analysis 67% of the time; however, given the language-specific nature of the heuristics, one would not necessarily

⁴⁰<http://search.cpan.org/perldoc/XML::Twig>

⁴¹<http://wiki.python.org/moin/PullDom>; <http://docs.python.org/library/xml.dom.pulldom.html>

expect a similar result for a transducer of another language. Alegria et al. (1996:200), in their Basque transducer, eliminate unlikely guesses using a slightly different heuristic based on the length and final segments of guessed stems; unlike Black et al., however, their software was not required to select one “best” analysis. In a test of 285 unknown words, the Basque guesser proposed the correct analysis (possibly among other analyses) 91.23% of the time. While the results for the English and Basque guessers cannot be directly compared, both are sufficiently high to support the conclusion that stem guessing is useful for extending the coverage of a morphological transducer.

2.4 Langgård and Trosterud’s Iñupiaq Transducer

To my knowledge, there exists only one other computational model of Iñupiaq: a proof-of-concept lexical transducer created by Per Langgård of Okaasileriffik (the Greenland Language Secretariat) and Trond Trosterud of the University of Tromsø (Langgård and Trosterud n.d.). The architecture of the transducer is based on Langgård’s much more extensive Greenlandic transducer and Trosterud’s transducers for Sámi. Langgård and Trosterud generously furnished me with the source code of their transducer, which I referred to regularly in the early stages of development of my own Iñupiaq transducer. This section presents an overview of the key features of their transducer, many of which I have incorporated into my own.

Langgård and Trosterud’s transducer is implemented with the Xerox Finite State Toolkit (XFST), described in Section 2.3 on pages 50–60.

2.4.1 Morphographemics

The transducer’s morphographemic component is implemented in *xfst* (see Section 2.3.3 on pages 53–54). It begins with the definition of a number of sets of graphemes and digraphs for convenience in writing rule contexts.⁴² These include a set of vowels, a set of stops, a set of nasals, etc., but also a set of rule triggers (referred to as “dummies” in Langgård and Trosterud’s source code; see Section 2.3.4 on pages 54–55) and a set of flag diacritics (see Section 2.3.5 on pages 55–58). Recall that both rule triggers and flag diacritics are treated as symbols in the transducer and that rules must take their presence into account (see pages 57–58).

The grapheme set definitions are followed by a number of morphographemic rules, which can be divided into three groups: those which operate due to rule triggers (used to model morphologically or lexically conditioned alternations), those which operate in the context of regular symbols (used to model phonologically conditioned alternations), and those which operate regardless of context (used to delete rule triggers and to convert other special symbols into standard orthographic symbols—for example, to convert the symbol {e}, used for /i/, into {i}). After all the rules have been defined, they are composed together in a single cascade.

⁴²I assume this is a common technique; it can also be seen in examples from Beesley and Karttunen (2003:e.g., 144, 470, 473) and Roark and Sproat (2007:71, 95).

2.4.2 Lexical Coverage

The transducer focuses primarily on Iñupiaq nominal and verbal morphology. Most stems were extracted from Webster and Zibell (1970). The bulk of the lexicon consists of noun and verb stems, with nine deverbal postbases (*-llatu-* ‘enjoy’, *-llatuniat-* ‘enjoy + FUT’, *-niaq-* FUT, *-niañit-* FUT+NEG, *-ñit-* NEG, *-saañi-* ‘deliberately’, *-saañiñit-* ‘not deliberately’, *-tiq-* ‘quickly, abruptly’, and *-vik* ‘place, time’), two denominal postbases (*-qaq-* ‘have’ and *-it-* ‘lack’), several verbal inflections (“present” and “past” indicative, interrogative, realized and unrealized contemporative, and consequential [only implemented for 3rd person singular intransitive]), unpossessed nominal inflections (absolutive, relative, locative, terminalis [not fully implemented], modalis [not fully implemented], and ablative [not fully implemented]) and two enclitics (*-lu* ‘and’ and *-guuq* ‘it is said that’). The lexicon also includes three pronouns (*suna* ‘what’, *kiña* ‘who’, and *una* ‘this (restricted and visible)’) implemented in the absolutive singular only. No demonstratives (other than *una*, just mentioned) or numerals are included. Some interjections are implemented, along with a few other particles. Several Malimiut stems are mixed in with the North Slope stems, as Webster and Zibell (1970) covers both dialects. Since the transducer does not contain Malimiut-specific phonological rules or inflectional endings, the inclusion of Malimiut stems is probably an error.

2.4.3 Lexical Architecture

2.4.3.1 Mechanics

The lexicon is implemented in *lexc* (see Section 2.3.2 on pages 51–53). The organization of the formative classes will be discussed in Sections 2.4.3.2 and 2.4.3.3 below.

A number of multicharacter symbols and other special symbols are used in the lexicon, particularly in the upper (analysis) language. These include flag diacritics, rule triggers, a number of grammar tags (e.g., *+N*, *+V*, *+Prs*, *+1Sg*, *+Abs*), and a morpheme boundary marker (*>*) which is used in the contexts of several morphographemic rules in addition to its obvious informative purpose in the analysis-side output. Postbases and enclitics in the lower language are also mapped to multicharacter symbols in the upper language (e.g., *+QAQ* for postbase *-qaq-* ‘have’, *+LU* for enclitic *=lu* ‘and’). This bears a certain similarity to the grammar tags traditionally used in FSTs since the publication of Karttunen et al. (1992) as upper-language representations of inflectional morphemes. But where grammar tags summarize an inflectional morpheme’s meaning, and if applicable, its place in a paradigm, these ‘postbase tags’ convey no such information. In fact, they convey no information beyond the citation form of the postbase or enclitic. Given that actual orthographic forms must be used in the lower language in any case, it seems simpler to use the same orthographic forms in the upper language. However, this is a minor contention and ultimately a matter of taste.

The orthography used in the transducer follows standard Iñupiaq conventions, with two exceptions:

- *{l}* (U+013E) is used in place of *{ł}* because the former is defined as a precomposed glyph in the Unicode standard, while the latter can only be represented as a sequence of two code points (U+0142 U+0323). Although the multicharacter symbol mechanism could be used to accommodate *{ł}*, it would require that rules be written with additional care. Also problematic is the fact that the com-

binning dot accent character is rendered improperly on many computers.

- Within the transducer, /i/ is represented as (e); however, this character is mapped back to (i) in all transducer output and so is not visible to end-users.

2.4.3.2 Morphologically Motivated Features

Langgård and Trosterud's lexicon is rather complex, not unlike the morphology it attempts to model. To simplify the discussion of their lexicon, I have omitted several less-significant details in favor of more clearly conveying the most important aspects of the design. For example, several formative classes in the lexicon exist primarily to add a particular multicharacter symbol to the appropriate lexical entries, such as tagging all nouns with +*N* or attaching a flag diacritic to the set of stems to which it pertains. While these are clever and appropriate uses of the formative class/continuation mechanism, they serve a relatively minor function and will be left out of the following discussion as well as Figure 2.15 on the next page. I have also taken the liberty of renaming some formative classes to make their functions more obvious. In reading this section and Section 2.4.3.3 below, bear in mind that the lexicon under review is a proof of concept and that in the course of further development, many or all of the issues raised below would likely be resolved.

A simplified overview of the lexicon appears in Figure 2.15 on the following page. As with the figures in Section 2.3.2, ϵ represents an empty string, a colon separates corresponding upper- and lower-language forms, and the symbol # marks the right boundary of a word in the lexicon. The string *TRUNC* is a rule trigger to cause deletion of a stem-final consonant. The symbol > serves as a morpheme boundary marker. Classes ending with *etc.* contain more members than are shown in the figure. Due to space constraints, some of the continuation classes which are referenced in the figure are not illustrated; these are marked with an asterisk. In all cases, classes with an asterisk are similar to some other class which is illustrated; for example, the *Irregularly Geminating Noun Suffixes* class, referenced in the *Noun Stems* class, is analogous to the *Default Noun Suffixes* class.

As discussed in Section 2.3.2, all lexicons designed in lexc begin with a root class. Langgård and Trosterud's root class branches to separate classes for noun stems (this class also includes particles), verb stems, and pronouns. With the exception of the particles, each member of the noun stem and verb stem classes continues to a class of suffixes; the specific class chosen depends in part on phonological factors, which will be discussed in Section 2.4.3.3 below. Particles and members of the pronoun class are complete words and continue directly to a class of enclitics (verbs continue to the enclitic class from classes of inflectional endings; nouns continue to the enclitic class from classes of inflectional and derivational morphemes). The enclitic class defines the right-side boundary of all strings in the lexicon; it contains an empty string member for words with no enclitics (which is the usual case).

Noun suffixes are separated into sets of singular, dual, and plural formative classes (postbases are included in the singular classes). This allows pluralia tantum (noun stems which cannot bear singular marking, such as *uniaq-*, stem of *uniat* 'sled') and singularia tantum (noun stems which can only bear singular marking; the only example of this in Langgård and Trosterud's transducer is *irri* 'cold air, weather') to be handled using continuation classes. Stems with no grammatical number restrictions continue to intermediate classes which branch to singular, dual, and plural marking (see *Default Noun Suffixes* in

Root	Pronoun Stems	Enclitics
ε <i>Noun Stems</i>	suná <i>Enclitics</i>	+GUUQ:>guuq #
ε <i>Verb Stems</i>	kiña <i>Enclitics</i>	+LU:>lu #
ε <i>Pronoun Stems</i>	una <i>Enclitics</i>	ε #

Noun Stems	
aaka	<i>Default Noun Suffixes</i>
aaqagu (<i>particle</i>)	<i>Enclitics</i>
amaquq	<i>Irregularly Geminating Noun Suffixes*</i>
apun:apute	<i>ti->n Noun Suffixes*</i>
irri:erre	<i>Default Singular Noun Suffixes</i>
siksrik:seksrek	<i>k-Final Noun Suffixes*</i>
uniat:uniaq	<i>Default Plural Noun Suffixes*</i>
<i>etc.</i>	

Default Noun Suffixes	Default Singular Noun Suffixes
ε <i>Default Singular Noun Suffixes</i>	+N+Abs+Sg:ε <i>Enclitics</i>
ε <i>Default Dual Noun Suffixes*</i>	+N+Rel+Sg:TRUNC>m <i>Enclitics</i>
ε <i>Default Plural Noun Suffixes*</i>	+QAQ:TRUNC>qaq <i>Default Intransitive Suffixes</i>
	<i>etc.</i>

Verb Stems	Default Intransitive Suffixes
kamik:kamek <i>Default Intransitive Suffixes</i>	ε <i>Nominalizers</i>
kamik:kamek <i>Default Transitive Suffixes</i>	ε <i>Verbal Derivatives</i>
kaviq <i>iC-Final Intransitive Inflections*</i>	ε <i>Default Intransitive Inflections</i>
nakuu <i>Vowel-Final Intransitive Suffixes*</i>	
tautuk <i>Vowel-Final Transitive Suffixes*</i>	
<i>etc.</i>	

Nominalizers	Verbal Derivatives
+VIK:>vik <i>k-Final Noun Suffixes*</i>	+LLATU TRUNC:>llatu <i>Vowel-Final Intransitive Inflections*</i>
	+LLATU TRUNC:>llatu <i>Vowel-Final Transitive Inflections*</i>
	+NIAQ:>niaq <i>Default Intransitive Inflections</i>
	+NIAQ:>niaq <i>Default Transitive Inflections*</i>
	<i>etc.</i>

Default Intransitive Inflections	Indicative Person Affixes
+V+Ind+Prs:>tu <i>Indicative Person Affixes</i>	+1Sg:>ŋa <i>Enclitics</i>
+V+Ind+Pst:>tua <i>Indicative Person Affixes</i>	+1Du:>guk <i>Enclitics</i>
+V+Int:>pi <i>Non-3rd Person Interrogative Intransitive Affixes*</i>	+1Pl:>tut <i>Enclitics</i>
+V+Int:>pa <i>3rd Person Interrogative Intransitive Affixes*</i>	+2Sg:>ten <i>Enclitics</i>
<i>etc.</i>	<i>etc.</i>

FIGURE 2.15: Simplified overview of the lexicon of Langgård and Trosterud's Inupiaq transducer

Figure 2.15), while stems which are restricted to specific grammatical numbers continue directly to the classes corresponding to the allowed numbers. One significant problem with this approach (at least as implemented in Langgård and Trosterud's transducer) is that it erroneously prevents derivational suffixation on pluralia tantum, since the stems must continue directly to number-marking suffixes.

Verb suffixes are divided into classes of nominalizing (deverbal-nominal) postbases, deverbal-verbal

postbases, and inflectional suffixes (with distinct sets of classes for intransitive and transitive endings). Intermediate formative classes are used to link most verb stems to multiple classes of suffixes (for example, the formative class *Default Intransitive Suffixes* in Figure 2.15 continues to a class of nominalizers, a class of verbal postbases, and a class of intransitive inflectional endings). For reasons of allomorphy, intransitive stems ending in /iC/ where C is a consonant other than /t/ continue directly to a formative class containing inflectional endings. A similar class designed as a continuation for stems ending in /it/ exists, but no stems are currently designated to continue to it. The phonological ramifications of these two formative classes will be discussed in Section 2.4.3.3 below. For the moment, I wish only to point out that this design, like the treatment of pluralia tantum discussed above, incorrectly disallows derivational suffixation on these verb stems.

All postbases in the lexicon continue to an appropriate class of inflectional endings. This prevents the transducer from computing words with more than one postbase (except for the few precomposed postbases Langgård and Trosterud have implemented, such as *-llatuniaq-* and *-saagijit-*; see Section 2.4.2 on page 61 for the full list of postbases in the transducer). This is a limitation of this particular transducer, not of the Iñupiaq language (see Section 2.1.4.3 on pages 36–37).

Langgård and Trosterud use flag diacritics (see Section 2.4.3.3 on pages 64–66) to enforce valence restrictions on verbal inflection. Two flags, IV and TV, are used. Stems are positively marked using the diacritic @P.IV.ON@ for intransitive stems, or @P.TV.ON@ for transitive stems. Inflectional endings are marked to require an “ON” value for the appropriate flag.

2.4.3.3 Phonologically Motivated Features

Several of the lexicon’s formative classes exist specifically to handle phenomena which are commonly treated as morphophonological in the Alaskan tradition of Eskimo linguistics. For example, noun stems ending in /t̃i/ whose absolutive singular form ends with ⟨n⟩ have a different continuation class than noun stems ending in, say, ⟨a⟩ or ⟨q⟩ (see Section 2.1.2.7 on pages 22–23). Where other continuation classes for noun stems have an empty string for the absolutive singular ending, the continuation class for ⟨n⟩-final nouns has the string *CVCTRUNCn* (*CVCTRUNC* is a rule trigger causing deletion of the stem-final (t̃i)). Similarly, verb stems ending in vowels have a different continuation class than verb stems ending in consonants, since many verbal inflections have different allomorphs depending on whether the stem-final segment is a consonant or a vowel (e.g., *pisuaq-tut* ‘they are walking’, *imiq-tut* ‘they are drinking’ vs. *tiji-rut* ‘they are taking off/flying away’, *nigi-rut* ‘they are eating’). For intransitive endings, additional continuation classes are provided for verb stems ending in /it/ (e.g., *tikit-chut* ‘they are arriving’) and stems ending in /iC/ where C is a consonant other than /t/ (e.g., *qitiġusiq-sut* ‘they are eating lunch’). These last two categories differ from the basic consonant/vowel distinction in that the forms which justify their existence (*-sut*, *-chut*, etc.) are not idiosyncratic allomorphs (as *-tut* and *-rut* are), but rather are derived from the consonant-following allomorphs (*-tut*, etc.) by productive phonological processes (specifically, palatalization; see Section 2.1.2.2 on pages 9–11). Langgård and Trosterud also handle “strong” consonants (see page 6)⁴³ and unpredictable gemination in nouns (see pages 17–18) in part through formative classes and

⁴³Per Langgård (personal communication, 22 March 2010) points out that the distinction between strong and weak consonants is not really phonological, and asserts that it is best treated as a distinction between two inflectional classes. However, as argued in Bills et al. (2010), it is not hard to treat the weak/strong consonant distinction as if it were phonological, in which case variant forms

continuations.

Creating separate formative classes for different phonological (or, in the case of strong and weak consonants, pseudo-phonological) phenomena has some drawbacks, not least of which is a high degree of redundancy between analogous classes. For example, the suffix *-qaq-* ‘have’ is defined, identically, in five classes: the default class, the class following noun stems with irregular geminates, the class following noun stems whose absolutive singular form ends in [n], the class following noun stems ending in /ġ/, and the class following noun stems ending in /k/. The absolutive singular first person singular possessive ending is defined identically in four of these five formative classes (it was omitted, probably unintentionally, from the class following noun stems ending in /ġ/). Within the verbal formative classes, the three classes of intransitive endings that follow verb stems ending in consonants (one class for stems ending in /it/, one for stems ending in /i/ followed by a consonant other than /t/, and one for stems ending in vowels other than /i/ followed by any consonant) are identical except for the entries for the indicative present and indicative past mood markers, which differ only in the first consonant.

The case of the verbal continuation classes for stems ending in consonants is reminiscent of the situation of French verbs whose penultimate vowel is a schwa (see Section 2.3.2 on pages 51–53). While those verbs could be modeled by creating separate formative classes for each ending (*-ecer*, *-emer*, *-eper*, etc.), it is simpler to use the general continuation class for verbs ending in *-er* and write rules to handle forms which differ from the vanilla *-er* paradigm. In the case of Iñupiaq verbs, the creation of separate continuation classes makes even less sense given that the rules responsible for the different forms (*-tut*, *-sut*, *-chut*, etc.) apply throughout the language, not just before verb inflections, so the transducer must incorporate them in any case.

A stronger case can be made for using separate formative classes for allomorphs such as *-tut* and *-rut* which do not result from synchronically productive phonological processes. It would be impractical to write rules for every such instance. The challenge in creating separate classes to deal with Iñupiaq allomorphy lies in the (relatively) large number of different conditioning environments to which various allomorphs are sensitive (see Section 2.1.2.8 on pages 23–25). The overlap between these environments necessitates the creation of several different classes, adding greatly to the complexity of the lexicon’s structure. For example, with allomorphs *-sima-* and *-ma-* ‘it is now known that’, selection depends on whether the preceding stem ends in a consonant or a vowel; with allomorphs *-uk-* and *-suk-* ‘want to’, selection depends on whether the stem ends in a back consonant or some other segment (see Section 2.1.2.8 on pages 23–25). To accommodate just these two morphemes, three continuation classes are required: one for stems ending in a vowel, which would contain (or continue to) *-ma-* and *-suk-*, one for stems ending in a back consonant, which would contain or continue to *-sima-* and *-uk-*, and one for stems ending in /t/, which would contain or continue to *-sima-* and *-suk-*. Some inflectional morphemes, such as the second person singular intransitive imperative, are sensitive to stem length and syllable structure (see 2.1.2.8), further compounding the problem.

Langgård and Trosterud’s transducer sidesteps many of these issues due to the limited number of suffixes it contains; its postbases exhibit little or no allomorphy, while its inflectional suffixes are slightly more complex. A more complete transducer would not have the luxury of avoiding these issues and would necessarily involve more complexity, either in the lexicon, the morphographemic rules, or both.

of inflectional endings can be treated as phonologically conditioned allomorphs rather than as forms conditioned by inflectional class.

While the structure of Langg ard and Trosterud’s lexicon seems unnecessarily complex, the additional complexity is deliberate (Per Langg ard, personal communication, 23 March 2010). By shifting more of the descriptive burden from morphophonological rules to enumerated forms in the lexicon, their design minimizes compile-time computation (and therefore the total time and resources required to compile the transducer). Put another way, many of the phenomena which could be handled computationally (for example, the generation of phonologically conditioned allomorphs) but would be recomputed every time the transducer was compiled, have instead been computed by hand, once and for all. The gains from this approach increase as the transducer increases in coverage, and particularly as more phonological phenomena are handled without rules. However, these gains are offset by the additional effort needed to create the lexicon (Bills et al. 2010:24).

2.4.4 File Structure and Compilation

The components of the transducer are spread over several files. The shell of the lexicon is in one file, with separate files for each of the following: noun stems, noun suffixes, verb stems, and verb suffixes. Most morphographemic rules are located in one file, while minor, optional rules for converting input words to lowercase and tokenizing text are located in separate files.

A UNIX Makefile contains the necessary instructions to combine all the files into a single transducer.

Chapter 3 Implementation

The final product of the work described in this thesis is a computer program which I refer to as the “Iñupiaq lexical transducer.” It is actually a pair of XFST-generated binary files: a strictly lexical transducer, and a stem-guessing transducer. These are intended to operate in sequence as described in Section 2.3.7 on pages 59–60.

The transducer is created by a handful of computer programs operating on a set of data files. This process is outlined in Figure 3.1. Primary data files are represented in bold, computer programs are in rectangles, and program outputs are italicized (I consider the morphographemic rules and instructions to build the transducer something of a hybrid between a primary data file and a computer program). The language or format of each file is given in parentheses. Languages and formats used include xfst and lexc (described in Section 2.3 on pages 50–60), Tcl (Tool Command Language; see Ousterhout and Jones [2009]), XML (Extensible Markup Language; see <http://www.w3.org/TR/REC-xml/>), and custom formats described later in this chapter.

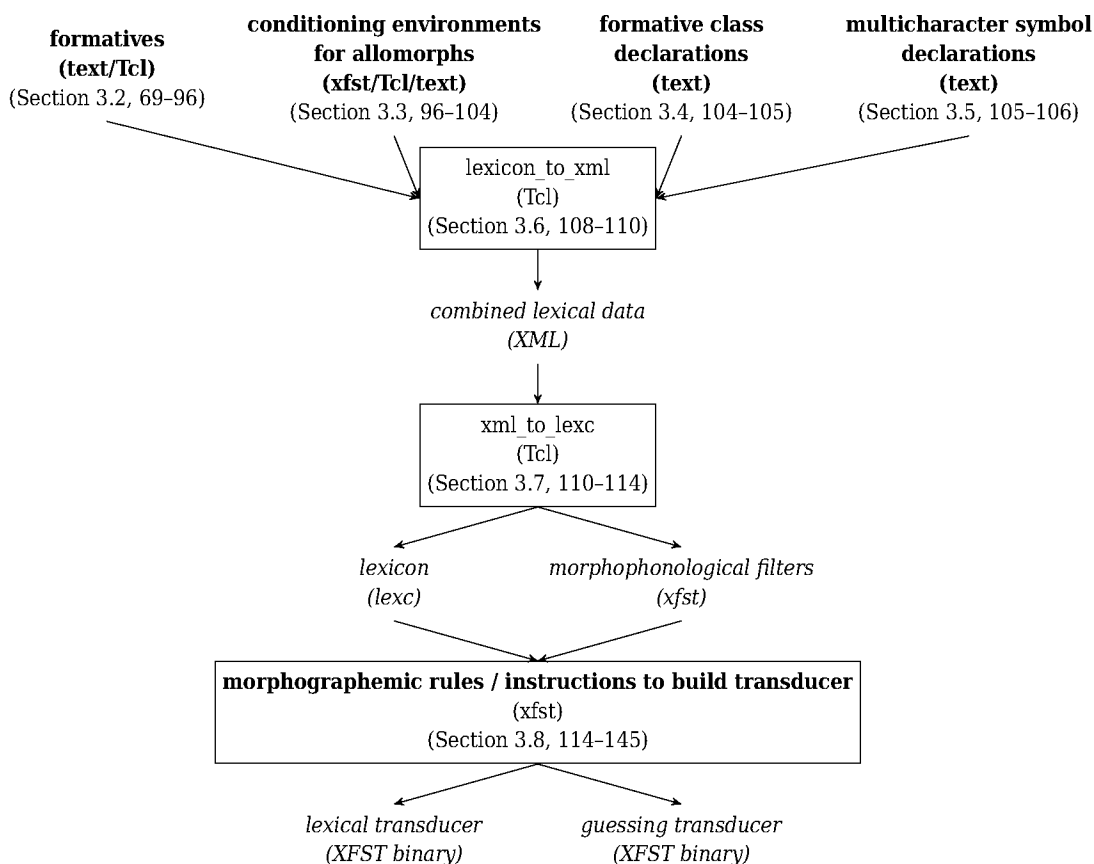


FIGURE 3.1: High-level schema of data and software used to produce the Iñupiaq transducer

3.1 Lexical Model

As can be seen in Figure 3.1 on the preceding page, the lexicon used in the Iñupiaq transducer is not specified directly in *lexc*. This fact allows the transducer to use a slightly different data model than the formative class/continuation model inherent in *lexc* (see Section 2.3.2 on pages 51–53), although the model inherits many concepts from *lexc*, including being built from formatives grouped together into classes.

Formatives in *lexc* possess the following properties: upper-language and lower-language representations; membership in a formative class; and a continuation to another class or to the end of the word. To these properties the Iñupiaq transducer adds properties for handling allomorphy and reduced forms (to be explained later), a way to specify an English gloss,¹ and the possibility to make a formative a member of more than one formative class. The complete set of formative properties in the Iñupiaq transducer model is given below (departures from the *lexc* model are italicized):

- a canonical form to be used as its upper-language representation
- *one or more allomorphs, each of which has:*
 - *a (lower-language) form*
 - *a conditioning environment*
- membership in one (or, in rare instances, *more than one*) formative class
- a continuation
- *an optional English gloss*
- *an optional reduced form (stems only)*

For inflectional endings, the canonical form will convey grammatical information such as grammatical category, number, case, or subject through the use of grammar tags (+N, +Pl, +Abs, +3Sg, etc.). For all suffixes, lower-language forms may include a suffix attachment pattern (see Section 2.1.2.6 on pages 18–22).

Formative classes in the Iñupiaq transducer serve similar purposes to their *lexc* counterparts, but the mechanisms by which they do so are conceptually different. In *lexc*, files are organized in terms of formative classes. Members of these classes are not limited to actual word formatives, but may also include “convenience” entries which serve to insert, for example, flag diacritics (see Section 2.3.5 on pages 55–58), rule triggers (see Section 2.3.4 on pages 54–55), or grammar tags into the lexicon. One particularly interesting class of non-linguistic entries is empty strings, which have no overt form but provide a continuation to some other formative class. In effect, these empty-string entries allow the class to which they belong to subsume, augment, or combine with the class to which they continue.² The notion of an entity whose sole function is to extend membership in one formative class to all members of another formative class is important enough to merit a name; I will refer to such an entity as an *epsilon continuation* (epsilon being the symbol conventionally used to represent empty strings in finite-state machines).

In the Iñupiaq transducer, lexical data is organized primarily in terms of individual formatives, rather than formative classes. Rather than treat flag diacritics (used to enforce valence and number limitations)

¹The gloss is not used in the transducer but may make the data files more readable, and may be useful if the lexicon is used in other projects.

²Subsuming, augmenting, and combining are not meant to be mutually exclusive operations, but rather three different ways to view the same operation.

and epsilon continuations as formatives, which clearly they are not, the Iñupiaq transducer treats them as properties of specific formative classes.

In addition to being grouped into classes, formatives in the Iñupiaq transducer are also categorized as stems, postbases, inflectional endings, or enclitics. Each category imposes a set of restrictions on the allowable properties of its members:

- Stems do not generally exhibit allomorphy (each stem is specified with a single allomorph whose conditioning environment is all-inclusive), and all stems in a particular formative class must have the same continuation class. No stem may belong to more than one class, although an equivalent effect can be achieved with epsilon continuations.³
- Individual postbases may not belong to more than one class.
- All inflectional endings in a particular formative class share a common continuation class.
- Enclitics do not exhibit allomorphy. All enclitics, regardless of the formative class to which they belong, share a single continuation class.

For purposes of imposing restrictions on specific categories as just described, reduced forms are considered stems, and post-inflectional derivational formatives are treated as postbases. The demonstrative prefix *ta(t)-* is considered part of the demonstrative stems.

3.2 Formative Data Files

Formative data for the Iñupiaq transducer’s lexicon is specified in four separate files, all in the data directory: *stems.txt*, *postbases.txt*, *inflections.txt*, and *enclitics.txt*. Rather than use *lexc*, which Beesley (2004a:2) has called a “dead end” (see Section 2.3.6 on pages 58–59), these files use a set of custom formats. A key advantage of these formats is ease of data entry. Recall that the *lexc* format requires formatives to be organized into formative classes that rarely correspond to the way linguists organize their data. In contrast to that approach, the formats used for this project allow stems, postbases, and enclitics to be specified in whatever order one wishes, including the order in which they appear in a dictionary. Inflectional data is specified in terms of tables, mimicking the way linguists generally structure paradigms. By freeing the computational morphologist from the need to drastically reorganize the data to be modeled, these formats make data entry much more straightforward.

The formats are text-based rather than XML-based; this is because writing XML by hand is cumbersome and error-prone. An anonymous reviewer of Bills et al. (2010) pointed out that XML editing software would adequately address these concerns, and in retrospect it would have been wise to store lexical data directly in XML. However, the formats created for this project are not entirely without merit; in particular, they are simple enough that

- they can be edited in any Unicode-compatible text editor,
- well-formedness and validity are trivially checked, and
- structure is minimal, but adequate to describe the appropriate data.

³To be more specific, one way to make stem σ a member of classes α and β , where β is the more restricted class, would be to include σ directly in β , and create an epsilon continuation from α to β .

At a later stage in the compilation of the Iñupiaq transducer, the lexical data entered using these formats is “uptranslated” to XML (see Section 3.6 on pages 108–110), so that it can be more easily used in other projects.

Because most of the lexical data comes from sources by Edna MacLean (1981; 1986*a*; 1986*b*; n.d.*b*; n.d.*a*), these files use an orthography based on her notational convention, which differs slightly from the standard Iñupiaq orthography in that “strong q” (*/q̄/*) is written ⟨Q⟩, and “strong i” (*/ī/*) is written ⟨I⟩ (the latter can optionally be written ⟨i⟩ in cases where surrounding letters distinguish it from “weak i”, for example, in a vowel cluster or preceding a palatalized consonant). One slight variation from MacLean’s orthography is used for the suffix *-m̄it-* ‘be located in, at, on ____’ and the various forms of the demonstrative *taīnna* ‘in that way or manner’, where the */ī/* occurs in a vowel cluster (see page 12); in these exceptional cases, */ī/* is represented by the symbol ⟨é⟩.

3.2.1 Stems

The file `data/stems.txt` contains all stems in the lexicon and associated metadata. It is divided into four sections: `Continuations`, `LongDistanceDependencies`, `GrammaticalCategoryTags`, and `Morphemes`. Entries in the `Morphemes` section define all the stems in the lexicon, as well as a handful of fully-inflected words. Each entry in this section is assigned a category code; category codes are used by the other sections to associate metadata with the stems. Section `Continuations` associates each category code with a continuation class; section `LongDistanceDependencies` assigns flag diacritics to certain category codes; and `GrammaticalCategoryTags` appends appropriate grammar category tags to upper-language representations of members of particle categories.⁴

3.2.1.1 Syntax

Each section begins with the word `SECTION` followed by a space and the name of the section. Data items in each section are separated by newline characters. Comments begin with an exclamation point and are ignored, as are blank lines.

Entries in the section `Continuations` consist of a category code followed by a space and the name of a continuation class (formative classes are defined in `data/classes.txt`; see Section 3.4 on pages 104–105). There should be exactly one entry in this section for each category code used in the file. Declaring continuation classes for unused category codes is not an error and has no effect, but failing to declare a continuation class for a category code which is used in the `Morphemes` section is an error.

The section `LongDistanceDependencies` is used to associate specific flag diacritics with category codes that involve long-distance dependencies (specifically, restrictions on inflectional valence for verbs and restrictions on grammatical number for certain nouns). Each entry begins with a category code followed by a space and a space-delimited list of xfst flag diacritics which apply to that category. The complete contents of this section are given in Figure 3.2 on the next page.

Note that verb valence is handled with positive flag diacritics while number restrictions are handled

⁴While not absolutely necessary, including grammatical category tags in analyses makes the transducer output more useful for applications such as automatic syntactic analysis. Tagging of uninflected grammatical categories happens in the stem component of the lexicon; tagging of inflected categories happens in the inflection component (see page 88).

```
SECTION LongDistanceDependencies
i @P.VALENCE.INTR@
t @P.VALENCE.TR@
it @P.VALENCE.INTR@ @P.VALENCE.TR@
n2+ @U.NUMBER.DU@ @U.NUMBER.PL@
n12 @U.NUMBER.SG@ @U.NUMBER.DU@
n1- @U.NUMBER.SG@
n2- @U.NUMBER.DU@
n3- @U.NUMBER.PL@
```

FIGURE 3.2: Contents of the LongDistanceDependencies section of the stem file

using unification flag diacritics.⁵

The section GrammaticalCategoryTags is used to append grammatical category tags to the end of upper-language representations of particles. All words in the upper language have a grammatical category tag; for inflected words, these tags are added after the inflectional ending and are defined along with those endings (see Section 3.2.3.2 on pages 87–92). Because particles are uninflected and not subject to postbase suffixation, grammatical tags can be added in the stem file. Entries in this section begin with a category code followed by a space and a grammatical category tag. (The tags begin with a percent sign to tell lex to treat the following greater-than sign, which is used in the transducer as a morpheme boundary marker, as a literal character.) The contents of this section are given in Figure 3.3.

```
SECTION GrammaticalCategoryTags
SECTION GrammaticalCategoryTags
conj %>+Conj
adv %>+Adv
interj %>+Interj
quest %>+Adv
```

FIGURE 3.3: Contents of the GrammaticalCategoryTags section of the stem file

The Morphemes section makes up the bulk of the file. Items in this section consist of a stem, a category code, a reduced form if applicable, an optional English-language gloss, and optional comments. Some examples are given in Figure 3.4 on the next page. All Iñupiaq stems cited in the next four paragraphs correspond to example entries given in this figure.

An entry minimally consists of a stem followed by white space followed by a category code, as with *pui-* ‘to emerge, surface’. Some entries in MacLean (1981) are made up of more than one word; these are arguably syntactic rather than morphological constructions and as such are not used in the transducer, but they are included in the stem file for completeness, as other applications may wish to use them. Multi-word lexemes are surrounded by curly braces, as with *napaaqtum aqargiq* ‘spruce grouse’. A stem may include separate upper- and lower-language representations; these are separated with a colon, as with

⁵For an explanation of flag diacritics, see Section 2.3.5 on pages 55–58. For flag diacritics associated with postbases, see pages 77–78. For flag diacritics associated with inflectional endings, see pages 86–87.

```

SECTION Morphemes
...
aasii conj ~asii
aiyugaaqĥiq i # to invite [someone/something] ! detransitivized form of aiyugaaq (2:91)
{napaaqtum aqargĥiq} n
ñiaq interj # don't do that
pui i
punniQ n ! student dictionary lists punniQ, but Kaplan's dissertation and MacLean's draft dictionary both list punniQ
qayuqYTCH n
su>+Pro+Abs+Sg:suna pro
tagĥruqHISTORICCONSONANT n
...

```

FIGURE 3.4: Example entries from the Morphemes section of the stem file

suna ‘what; what thing’. Some stems are subject to lexically conditioned morphophonological events, particularly irregular gemination (see pages 17–18); the lower-language representation of these stems ends with a multicharacter rule trigger symbol such as YTCH or HISTORICCONSONANT (see specification of *qayuq* ‘broth’ and *tagĥiuq* ‘salt; ocean’ in Figure 3.4).

If a stem can occur as a reduced form (see on pages 34–36), the surface form of the reduced form is given after the category code and marked with a tilde, as with *aasii* ‘and [then]’. The upper-language form of the stem will also be used for the reduced form, so that a word like *iglarraqsivĥuniasii* ‘and then he began laughing’ (MacLean 1986b:10) will produce the analysis *iglaq>rraqsi>+V+Cont1+Real+3Sg|aasii>+Conj* with the final formative analyzed as *aasii* rather than *asii* (the pipe symbol [|] marks the left boundary of a reduced form).

English-language glosses are optional; they were not originally part of the file format but were added when data from Webster and Zibell (1970) was added because glosses from that dictionary were already available in electronic format and did not need to be re-entered. For the most part, glosses have not been added to entries other than those from Webster and Zibell, although this could be done if the need arose. Glosses begin with a hash mark (#) and optional whitespace. Glosses follow the category code and reduced form (if any); see *ñiaq* ‘don’t do that!’.

Comments are also optional and begin with an exclamation point. These must occur to the right of any meaningful data (see *punniĥ*). They are strictly for the benefit of those reading and editing the file and are completely ignored by the computer. An entry may of course contain a gloss and a comment, as with *aiyugaaqĥiq*.

3.2.1.2 Category Codes

As was seen in the previous section, category codes are used to assign continuation classes to items, to impose long-distance dependency constraints as appropriate, and to add grammatical category tags to the upper-language forms of particles. The list of category codes used in the stem file is given in Table 3.1 on the following page.

Some of the categories in Table 3.1 deserve some additional explanation. Category ‘*’ is used in cases

TABLE 3.1: Stem category codes and associated metadata

Code	Explanation	Continuation class	Other properties
n	nouns	NounSuffixes	
i	intransitive verbs	VerbSuffixes	intransitive valence flag diacritic
it	ambitransitive verbs	VerbSuffixes	transitive and intransitive valence flag diacritics
t	transitive verbs	VerbSuffixes	transitive valence flag diacritic
interj	interjections	EncliticsOrEnd	grammar tag >+Interj
conj	conjunctions	EncliticsOrEnd	grammar tag >+Conj
adv	adverbs	EncliticsOrEnd	grammar tag >+Adv
*	inflected words with morphological information included directly in the entry	EncliticsOrEnd	
N	proper nouns	NounSuffixes	
da	demonstrative adverbs	DemAdvSuffixes	
dp	demonstrative pronouns	DemProSuffixes	
dpna	irregular forms of demonstrative pronouns that do not occur in the absolutive case		
dpns	irregular forms of demonstrative pronouns that do not occur in the singular	DemProNonAbsSuffixes	
n12	noun stems which occur only in singular and dual	NounSuffixes	singular and dual number flag diacritics
n1-	noun stems which occur only in singular (singularia tantum)	NounSuffixes	singular number flag diacritic
n2+	noun stems which occur only in dual and plural (pluralia tantum)	NounSuffixes	dual and plural number flag diacritics
n2-	noun stems which occur only in dual	NounSuffixes	dual number flag diacritic
n3-	noun stems which occur only in plural	NounSuffixes	plural number flag diacritic
num	number stems	NumSuffixes	
pos	positional noun stems	PositionalBaseSuffixes	
pro	personal pronouns	EncliticsOrEnd	
kisi	forms of the pronoun <i>kisi</i> 'X alone'	KisiSuffixes	
quest	interrogative particles	EncliticsOrEnd	

where an entire inflected word, rather than just the stem, is included in the stem file. This is done in cases where the inflection differs enough from the standard pattern that it is as easy or easier to enumerate the forms explicitly than to create special continuation classes to handle the situation. For example, *inna* and *tainna* may be treated as “defective” demonstrative adverbs; in addition to their basic (“interjectional”) form (see on page 27), they may occur in terminalis and similaris cases, but not in vialis or ablative (MacLean 1986b:53–54). Rather than create a special inflectional continuation class for these two stems, I just list all inflected forms in the stem file:

inna>+Dem+Adv:enna *
 tat>inna>+Dem+Adv:taenna *
 inna>+Dem+Adv+Trm:ennamun *
 tat>inna>+Dem+Adv+Trm:taennamun *
 inna>+Dem+Adv+Sim:ennatun *
 tat>inna>+Dem+Adv+Sim:taennatun *

Another set of words included in category ‘*’ are irregular first person possessed relative forms of kinship terms: *aaka-a* ‘(of) my mother’, *aapa-a* ‘(of) my father’, *aana-a* ‘(of) my grandmother/great-aunt’, and *ataataa* ‘(of) my grandfather’. The stems of these nouns are otherwise regular and are listed separately in the stem file as normal nouns, but these four irregular forms are listed in their full forms:

aaka>+N+Rel+Sg+1Sg:aakaa *
 aapa>+N+Rel+Sg+1Sg:aapaa *
 aana>+N+Rel+Sg+1Sg:aanaa *
 aataata>+N+Rel+Sg+1Sg:aataataa *

Categories ‘da’ (containing demonstrative adverbs) and ‘dp’ (containing demonstrative pronouns) differ from most other categories in that the listed forms are not stems but complete citation forms. Theoretically, demonstrative stems fall into a single category, but there can be considerable differences between the adverbial and pronominal manifestations of those stems, as can be seen in Table 3.2 below (adverbs in the table are given in “interjectional” (citation) form; pronouns are given in nominative singular). Since most non-linguists are unaccustomed to demonstrative stems, using citation forms for the upper-language representations makes the transducer easier for a larger audience to use. Within the transducer, morphographic rules derive the stems from the citation forms (see Section 3.8.2.12 on pages 126–128).

TABLE 3.2: Demonstrative stems realized as adverbs and pronouns

Stem	Adverb	Pronoun	Meaning
<i>uv-</i>	<i>uvva</i>	<i>una</i>	object or area that is restricted, visible, and near to the speaker
<i>mar-</i>	<i>marra</i>	<i>manna</i>	object or area near the speaker, extended, and visible
<i>qakim-</i>	<i>qaqma</i>	<i>qakimna</i>	object or area outside or next door, distant from both speaker and listener, and not visible

The odd category ‘dpna’ (demonstrative pronoun, no absolutive) exists specifically for a variant of the pronoun *igña* denoting a restricted, visible object or area away from both speaker and listener, down the coast to the west or across a river or large area. Most demonstrative pronouns have dialectal variants for inflected singular forms in cases other than absolutive, and these variants can be reliably generated by morphographic rules (see page 128); but *igña* has a set of variants beginning with *irr-* which are not generated by those rules. Rather than create special rules to generate these forms, I define the stem entry *igña:irr dpna*, as well as a special continuation class ‘DemProNonAbsSuffixes’ containing all singular

non-absolute inflectional endings for demonstrative pronouns (see Section 3.2.3.1 on pages 84–87). An alternative to this approach would have been to create categories similar to ‘*’ but having the same continuation classes as inflected demonstrative pronouns, and to explicitly enumerate all forms of *igña* that begin with *irr-*. Because demonstrative continuation classes are somewhat complex, however, the current approach is probably easier.

The category ‘dpns’ (demonstrative pronoun, no singular) has a similar *raison d’être*. Demonstrative pronoun *manna* has predictable dual and plural forms beginning with *matk-*. But variant forms beginning with *mak-* and *makk-* also exist, and the rule described page 128 will not generate these. To deal with this situation, the stem file contains the entries *manna:ma dpns* and *manna:mak dpns*. The continuation class for these special stems is ‘DemProNonSingularSuffixes’; an instruction in the inflectional ending file populates this class with only non-singular demonstrative pronoun inflectional endings (see Section 3.2.3.1 on pages 84–87).

A number of categories exist specifically to limit the grammatical numbers for which particular nominal stems can be inflected: ‘n12’, ‘n1-’, ‘n2+’, ‘n2-’, and ‘n3-’. All of these are used at least once except ‘n12’ and ‘n1-’; the fact that they are defined but not used has no impact on the transducer. I treat limitations of grammatical number as a long-distance dependency similar to limitations on verb valence, and use the flag diacritic NUMBER to enforce this restriction (see Section 2.3.5 on pages 55–58). Many members of the ‘n2+’ category are specified with an upper-language form identical to the (dual or plural) citation form of the stem, and with a lower-language form without this number inflection. For example, the entry for *kamiktuuk* ‘pants’ is *kamiktuuk:kamiktuk n2+*. The uninflected stem, *kamiktuk*, is used within the transducer, producing correct suffixed forms (for example, *kamiktuqtuq* ‘he is wearing pants’), but for purposes of specifying or generating an analysis, the more familiar form *kamiktuuk* is given even though this bears dual inflection. Members of the categories ‘n2-’ and ‘n3-’ are specified in a similar manner.

Category code ‘num’ is used for number stems such as *atausiq* ‘one’ and *malguk* ‘two’. Numbers in Iñupiaq are grammatically nouns, but there are a handful of postbases which attach only to number stems. Having a separate category for numbers makes it possible to ensure that both general nominal postbases and number-specific postbases may be suffixed to number stems.

Category code ‘pro’ is used for personal pronouns. All forms of personal pronouns are explicitly enumerated in the stem file. The basic personal pronouns are given with upper-language forms consisting entirely of grammatical tags; for example, *uvaŋa* ‘I’ is specified as *+Pro+1Sg+Abs:uvaŋa pro*; *iliŋiññun* ‘to them (3 or more)’ is specified as *+Pro+3Pl+Trm:iliŋiññun pro*. Also included in category ‘pro’ are forms of *kiña* ‘who?’, *kisu* ‘which one?’, *iluqaq* ‘all of X’ (e.g. *iluqaqma* ‘all of me’; *iluqaïsa* ‘all of them [3 or more]’), and *tamaq* ‘all of X’ (e.g. *tamaqma* ‘all of me’; *tamaïsa* ‘all of them [3 or more]’). Forms of the stem *kisi* ‘X alone’ are also pronominal, but are given the category code ‘kisi’ to allow them to continue to the enclitic *=tchiaq* ‘by X’s self’ (Edna MacLean, personal communication, 2 July 2009). Example entries are given below:

Example entry	Gloss
kiña>+Pro+Sg+Abs:kiña pro	‘who?’
kiña>+Pro+Pl+Abl:kitkunniñ pro	‘from whom (3 or more)?’
kisu>+Pro+Du+Abs:kisuk pro	‘which two?’
kisu>+Pro+Sg+Abl:kisumiñ pro	‘from which one?’
iluqaq>+Pro+1Sg:iluqaqma pro	‘all of me’
iluqaq>+Pro+3Pl+Trm:iluqaqannun pro	‘to all of them’
tamaq>+Pro+1Sg:tamaqma pro	‘all of me’
tamaq>+Pro+3Pl+Trm:tamaqannun pro	‘to all of them’
kisi>+Pro+3RSg:kisiml kisi	‘he alone’
kisi>+Pro+3Sg:kisian kisi	‘him alone’
kisi>+Pro+2Du+Abl:kisivsigñiñ pro	‘from the two of you alone’

Forms of *kiña* and *kisu* are not marked for person; the person of *kiña* is by definition unknown, and *kisu* can only be 3rd person. Forms of *iluqaq*, *tamaq*, and *kisi* are not marked for case except in terminalis or ablative. They cannot occur in modalis, similaris, vialis, or locative case, and it would be inappropriate to label their default form as either absolutive or relative; in this form, they may function as subjects or objects of transitive verbs or subjects of intransitive verbs (in the 3rd person, a distinction is made between “reflexive” and “non-reflexive” forms; non-reflexive forms are used as objects of transitive verbs, while reflexive forms are used as subjects of transitive or intransitive verbs).

3.2.1.3 Sources and Data Entry Methodology

Stem data in the transducer comes from three main sources: MacLean (1981, n.d.a) and the electronic version of Webster and Zibell (1970). Additionally, MacLean (1986a, b, n.d.b) were mined for stems. Of these, by far the most comprehensive is MacLean (n.d.a), but permission to use this resource was not secured until after the other stem data had already been entered; otherwise, I probably would not have bothered with the other material. There is a certain amount of overlap between the lexical coverage of the sources used, but while this has made data entry less efficient, it has not had any detrimental effects on the transducer itself.

Stems from MacLean (1981) were typed by myself as well as Carnegie Mellon University undergraduates Ida Mayer, J. Eliot DeGolia, and Sai Venkateswaran and University of Pittsburgh undergraduate Paul Lundblad; later, I tagged each stem with an appropriate category code. The rest of the data entry was carried out by me. The content of Webster and Zibell (1970) was extracted from the Internet using a Tcl script and converted into a spreadsheet, where entries marked as Kobuk (Malimiut) forms were filtered out, inflectional endings manually removed from words, instances of (i) and (q) marked as strong or weak when such distinctions could be made (stems for which a distinction could not be made were omitted from the lexicon); and category codes added. Stems from the end-of-chapter vocabulary lists in MacLean (1986a, b, n.d.b) were added directly to the stem file together with appropriate category codes. Similarly, stems from MacLean (n.d.a) were listed and categorized directly in the stem file.

For most stems, it was clear which category code to assign. The overwhelming majority of stems in the lexicon are nominal or verbal, and the dictionary definitions of most verbal stems give a clear indication of

the stem's valence. I have classified stems that did not seem to fit into any other category as adverbs, and while I think this characterization is mostly accurate, the adverb category probably deserves a thorough review at some future time.

3.2.2 Postbases

Postbases and post-inflectional derivational suffixes are defined in the file `data/postbases.txt`. The file is structured similarly to `data/stems.txt`, but must take into account the specific properties of postbases:

- postbases must be categorized along two axes: the class of stems to which they attach, and the class of stems which they derive
- many postbases exhibit phonologically-conditioned allomorphy; each allomorph needs its own lower-language form, but all allomorphs should map to the same upper-language form; also needed is a description of the conditions under which each allomorph surfaces
- some postbases will be subject to valence constraints imposed by stems (for example, attaching only to intransitive stems); others will impose their own long-distance constraints (for example, valence changers and denominal postbases with an inherent grammatical number) and still others will neither impose nor be subject to any particular valence or grammatical number constraints

To account for these facts, the sections `LongDistanceDependencies` and `Morphemes` are structured slightly differently from their counterparts in the stem file. There is also an additional section, `Categories`. A key difference from the stem file is that entries in the `Morphemes` section are given two category codes: an attaching category code defining the type of stem to which they may attach, and a resulting category code defining the type of stem the postbase derives.

3.2.2.1 Syntax

As with the stem file, sections begin with the word `SECTION` followed by a space and the name of the section; items in each section are separated by newline characters; and comments (which begin with an exclamation point) and blank lines are ignored.

The section `Categories` associates each attaching category code with a specific formative class. Each attaching category code used in the `Morphemes` section should correspond to exactly one entry in the `Categories` section. Each entry consists of an attaching category code followed by a space and the name of the corresponding formative class. The contents of this section are given in Figure 3.5 on the following page.

The section `Continuations` associates each resulting category code with a specific continuation class. The format is exactly the same as that of the `Categories` section. The contents of this section are given in Figure 3.6 on the next page.

The section `LongDistanceDependencies` is quite different from its stem-file counterpart. This is because postbases may impose long-distance dependencies based on either the attaching category code, the resulting category code, or both.

Entries in this section consist of an expression followed by a space followed by a space-delimited list of flag diacritics. The expression is surrounded in curly braces and consists of one or more triples of the

```

SECTION Categories
n NominalPostbases
num NumberPostbases
i VerbalPostbases
t VerbalPostbases
it VerbalPostbases
pos PositionalBasePostbases
DemAny DemAnyPostInflection
DemAdvAbl DemAdvAblPostInflection
DemAdvTrm DemAdvTrmPostInflection
DemAdvVia DemAdvViaPostInflection
DemAdvLoc DemAdvLocPostInflection
DemProLoc DemProLocPostInflection
NTrm TerminalisNounPostInflection
NLoc LocativeNounPostInflection

```

FIGURE 3.5: Contents of the Categories section of the postbase file

```

SECTION Continuations
n NounSuffixes
n2- NounSuffixes
n3- NounSuffixes
i VerbSuffixes
t VerbSuffixes
it VerbSuffixes
same VerbSuffixes
Cont1 Cont1Inflection
ContNeg ContNegInflection
Cond CondInflection
Conseq ConseqInflection
* EncliticsOrEnd

```

FIGURE 3.6: Contents of the Continuations section of the postbase file

form <CODE> <COMPARISON OPERATOR> <VALUE>; if an expression contains more than one triple, they are joined by logic operators && (logical and) or || (logical or). <CODE> is either “CATEGORY” (the attaching category code) or “CONTINUATION” (the resulting category code). <COMPARISON OPERATOR> is either “eq” (equals), “ne” (does not equal), “in” (is a member of the following list), or “ni” (is not a member of the following list). If <COMPARISON OPERATOR> is “eq” or “ne”, then <VALUE> should be a category code enclosed in double quotes (“”); if <COMPARISON OPERATOR> is “in” or “ni”, then <VALUE> should be a space-delimited list of category codes enclosed in curly braces.

Each expression defined in the LongDistanceDependencies section is evaluated against each entry in the Morphemes section. If the expression evaluates to true, then the flag diacritics associated with that expression are applied to the postbase defined in the Morphemes section entry. The contents of the LongDistanceDependencies section are shown in Figure 3.7 on the following page.

The Morphemes section contains all the postbases defined in the transducer. Entries in this section

```

SECTION LongDistanceDependencies

! all deverbal postbases clear valence
{CATEGORY in {i t it} && CONTINUATION ni {i t it same}} @C.VALENCE@

! dual-only postbases require dual inflection
{CONTINUATION eq "n2-"} @U.NUMBER.DU@

! plural-only postbases require plural inflection
{CONTINUATION eq "n3-"} @U.NUMBER.PL@

! intransitive-only postbases require an intransitive stem
{CATEGORY eq "i"} @R.VALENCE.INTR@

! transitive-only postbases require a transitive stem
{CATEGORY eq "t"} @R.VALENCE.TR@

! postbases that create verbs need to set appropriate transitivity flags
{CONTINUATION eq "i"} @P.VALENCE.INTR@
{CONTINUATION eq "t"} @P.VALENCE.TR@
{CONTINUATION eq "it"} @P.VALENCE.INTR@ @P.VALENCE.TR@

```

FIGURE 3.7: Contents of the LongDistanceDependencies section of the postbase file

consist of a form specification, a space, an attaching category code, another space, a resulting category code, an optional English-language gloss and an optional comment. Category codes will be explained in greater detail in Section 3.2.2.2 on pages 80–83. Glosses begin with a hash mark and continue until the beginning of a comment or the end of the line. Comments begin with an exclamation point and continue to the end of the line; they are ignored in the process of building the lexicon.

The form specification may be quite complex. In its simplest form, it consists of a suffix attachment symbol followed (with no intervening space) by the orthographic form of the postbase (in MacLean’s orthography) which will serve as both the upper-language and lower-language representation of the postbase. The suffix attachment symbols are based on the ones used by Edna MacLean (see Section 2.1.2.6, particularly Table 2.4 on page 19). The symbols used in the transducer are given in Table 3.3.

TABLE 3.3: Suffix attachment symbols used in the transducer (see also Table 2.4)

Transducer symbol	Corresponding symbol in MacLean system
-	—
+	+
:	:
/	÷
+-	±
-+	∓
=	=
~	~

If a postbase has multiple allomorphs, the form specification is enclosed in curly braces and consists of space-separated pairs of conditioning pattern names and allomorph forms. Conditioning patterns are defined in `data/patterns.txt` (see Section 3.3 on pages 96–104). The first allomorph form given is used as the upper-language representation for all allomorphs.

Shorthand annotations exist for some common forms of allomorphy. If the initial consonant alternates depending on whether the preceding phoneme is a consonant or a vowel, the consonant-following phoneme can be given, followed by two forward slashes and the vowel-following phoneme, followed by the non-alternating part of the morpheme: `-+t//liq` (`-tiq/-liq` ‘quickly, abruptly’). If a postbase begins with a consonant only when the preceding phoneme is a vowel or `/t/`, that consonant may be surrounded by square brackets: `+ [s]uk` (`-suk/-uk` ‘to want to’). Both of these conventions come directly from MacLean (1981).

In rare cases, it’s necessary to specify an upper-language form for a postbase that is different from the lower-language form. This is necessary for post-inflectional suffixes, where the upper-language form contains grammatical tags, which should not appear in a surface form. It is also done for postbase `-gaaḡruit` ‘many’ whose citation form includes the absolutive plural ending `-it`, which must be removed from the surface form so that other inflectional endings (say, the ablative plural) may be correctly suffixed to it. A comma is used to specify separate upper- and lower-language forms; there must be no space before or after the comma. Only the first allomorph in a list can have separate upper- and lower-language forms.

Example postbase entries are given in Figure 3.8.

Example entry	Gloss
<code>-ḡaaḡuit,-ḡaaḡuk n n3-</code>	many
<code>+ḡruiññaQ n n</code>	merely, only, just a ____
<code>+N+Via+Sg%>kuaq,-'kuaq n i</code>	to go by way of a ____ (which is singular)
<code>{?V +pkaq ?C -+tit} it it</code>	(intrans.) to cause or allow oneself to be ____ed; (trans.) to allow or cause him/it to ____, be ____ed, or remain ____
<code>+ [s]uk it same</code>	to want to
<code>-+t//liq it same</code>	quickly, abruptly

FIGURE 3.8: Example entries from the Morphemes section of the postbase file

3.2.2.2 Category Codes

The complexity of the postbase file having two sets of category codes is offset somewhat by the fact that the number of codes in the two sets combined is smaller than the number of codes defined in the stem file. The complete set of codes is shown in Table 3.4 on the following page.

TABLE 3.4: Postbase category codes

Attaching category codes

Code	Explanation
n	noun-suffixing postbases
num	number-suffixing postbases
i	postbases suffixing intransitive verbs
t	postbases suffixing transitive verbs
it	postbases suffixing verbs of either valence
DemAdvAbl	post-inflectional suffixes for demonstrative adverbs in ablative case
DemAdvTrm	post-inflectional suffixes for demonstrative adverbs in terminalis case
DemAdvVia	post-inflectional suffixes for demonstrative adverbs in vialis case
DemAdvLoc	post-inflectional suffixes for demonstrative adverbs in locative case
NTrm	post-inflectional suffixes for nouns in terminalis case

Resulting category codes

Code	Explanation
n	de-nominal suffixes
n2-	inherently dual de-nominal postbases
n3-	inherently plural de-nominal postbases
i	inherently intransitive de-verbal suffixes
t	inherently transitive de-verbal suffixes
it	inherently ambitransitive de-verbal suffixes
same	de-verbal suffixes with no inherent valence
*	suffixes which do not permit additional derivation or inflection

Attaching category codes ‘DemAny’, ‘DemAdvAbl’, ‘DemAdvTrm’, ‘DemAdvVia’, ‘DemAdvLoc’, ‘DemProLoc’, ‘NTrm’, and ‘NLoc’ are used for post-inflectional suffixes and apply to only a few suffixes, given in Figure 3.9 (see pages 30–34).

```
-aglaaq NTrm i
OPTGEMaq DemAdvVia i
+mlk DemAdvAbl i
+mlñ DemAdvAbl *
+nmun DemAdvTrm *
+nmuk DemAdvTrm i
+q DemAdvAbl DemAdvTrm i
-qpanl DemAdvLoc *
-qpaniñ DemAdvLoc *
-qpanun DemAdvLoc *
+qsiuq {NLoc DemAdvLoc DemProLoc} i
{?Always {saagrúk%>+Adv,-saagrúk saaq%>+Adv,-saaq}} DemAny i
```

FIGURE 3.9: Post-inflectional suffixes implemented in special categories

Most nominal post-inflectional suffixes are given a slightly different treatment (see listing in Figure 3.10). Unlike the suffixes in Figure 3.9, which are strictly separated from the inflectional endings they follow, the nominal suffixes are encoded as compounds of an inflectional ending and a post-inflectional suffix, and categorized with attaching category code ‘n’ like more typical noun-suffixing postbases. In their upper-language forms, they contain both grammatical tags (indicating the case from which the suffix was derived and the grammatical number it conveys) and an orthographic form which is a blend of the singular inflectional ending and the following suffix. In their lower-language forms, they consist of a blend of the actual ending, whatever its grammatical number, and the following suffix. The inclusion of case tags in the upper-language representation of these suffixes is unorthodox, but provides consistency with the treatment of derivational post-inflectional suffixes. The use of a blended citation form reflects the treatment of these suffixes in MacLean (1986*a*), but creates some inconsistency with demonstrative post-inflectional suffixes. However, this inconsistency comes from the divergent treatment of these two sets of suffixes in MacLean (1986*a, b*); since students and scholars of Iñupiaq are most likely to associate these suffixes with the citation forms given in these two books, I believe it makes sense to reproduce them using these forms, consistent or not.

```

+N+Loc+Sg%>mlet,/mlet n i
+N+Loc+Du%>mlet,ABSDUAL/nlet n i
+N+Loc+Pl%>mlet,/nlet n i

+N+Abl+Sg%>miñḡaq,/miñḡaq n i
+N+Abl+Du%>miñḡaq,ABSDUAL/niñḡaq n i
+N+Abl+Pl%>miñḡaq,/niñḡaq n i

+N+Trm+Sg%>muk,/muk n it
+N+Trm+Du%>muk,ABSDUAL/nuk n it
+N+Trm+Pl%>muk,/nuk n it

+N+Via+Sg%>kuaq,'-kuaq n i
+N+Via+Du%>kuaq,ABSDUAL/kuaq n i
+N+Via+Pl%>kuaq,/tiguaq n i

```

FIGURE 3.10: Post-inflectional suffixes implemented as “hybrids” within the standard nominal suffix formative class

Resulting category codes ‘n2-’ and ‘n3-’ apply only to a handful of postbases, such as *-giik* ‘a relationship where one has another as his/her/its ____’ (MacLean n.d.*a*) and *-ḡaaḡuit* ‘many’.

Resulting category codes ‘i’, ‘t’, and ‘it’ involve implicit valence. Most postbases categorized with resulting category code ‘it’ are noun-suffixing deverbal postbases such as *-iñḡiḡuq-* ‘to need or experience a lack of ____ (for him/her/it/etc.); to have difficulty providing ____ for him/her/it/etc.’ and *-tchiaq-* ‘to acquire a new ____ (for him/her/it/etc.)’. Valence-increasing suffixes such as *-tit-/pkaq-* (causative) are also currently categorized this way to allow for intransitive inflection (which indicates that the direct object is co-referential with the subject). Since this property is shared by transitive verbs generally, this is probably a design flaw.

Resulting category code ‘same’ is used with verbal postbases to indicate no change in valence. It is used with postbases such as *-anik-* ‘previously, already’, *-kaṇîṭ-* ‘to delay ____ing, not ____ early’, and *-(si)ma-* ‘it is now known that’.

3.2.2.3 Sources and Data Entry Methodology

Initially, postbase data was drawn from MacLean (1981, 1986*a, b*, n.d.*b*), with a handful of additional postbases drawn from Webster and Zibell (1970). When permission was secured to use MacLean (n.d.*a*), an entirely new postbase file was created containing only material from this dictionary, with the exception of five “post-inflectional” demonstrative suffixes from MacLean (1986*b*:ch. 16), which are defined in the dictionary but were copied from the previous postbase file where they were conveniently grouped together. Other than these five suffixes, no material from the previous postbase file is currently used in the transducer.

Allomorphs and suffix combination patterns were checked against examples given in the dictionary and adjustments made as necessary to ensure that each postbase would generate the proper forms in combination with the transducer’s morphographemic rules. For example, a number of postbases in the dictionary begin with the string *+ [g]*, for example *+ [g]i-* ‘to have him/her/it for one’s ____’; *+ [g]iit-* ‘to have a bad ____; to have a ____ache’; *+ [g]Ik-* ‘to be tasty, beautiful, handsome; to have a beautiful, good ____’. When these suffixes attach to a stem ending in a single consonant, the resulting word contains a single consonant at the boundary of the two morphemes: ⟨g⟩ if the stem ended in ⟨t⟩ or ⟨k⟩, and ⟨ġ⟩ if the stem ended in ⟨q⟩. Entering these postbases using the notation *+ [g]* will not yield these results, so instead the postbases are entered with two allomorphs, one that begins with *-ġ* which occurs following stems ending in ⟨q⟩, and another that begins with *{-g}* and occurs in all other environments.

Postbases marked “limited” in MacLean (n.d.*a*) were not entered into the postbase file, because they represent historic postbases whose only use now is in lexicalized forms (which would be listed in the stem file). Allomorphs listed with combination patterns = or ~ were almost always excluded from the file because these patterns are mostly unproductive and generate large numbers of incorrect analyses when associated with short suffixes. Currently, only three allomorphs in the transducer are specified with the = pattern: *-îlaq(-)* ‘one without, area which lacks ____; one who is not ____; to be or do without ____’; *-îqî-* (variant of *-lîqî-*) ‘having to do with hunting, preparing, repairing, cleaning, associating, messing around with ____’; and *-uraq* ‘a smaller version of a ____; a diminutive ____; immediate vicinity of ____’. Only one allomorph is specified with the ~ pattern: *-qsraq-* ‘to continue to ____; to experience ____’; its application is limited to stems ending in */i/*.

3.2.3 Inflectional Endings

The inflectional ending file is quite different from all other formative files, which is fitting given that inflectional endings are quite different from all other formatives. In reference works, stems, postbases, and enclitics are presented in list format, as dictionary or glossary entries. In contrast, inflectional endings are given in tabular format, in paradigms. As lexc is inherently list-like, the original purpose of the inflectional ending file (or, more accurately, its predecessor) was to generate a list of inflectional endings from tabular data. The tabular data was originally intended to be temporary, and the generated list was to become

the permanent source of inflectional data. However, it became clear that the tabular format was a more natural treatment for these formatives, and it was retained, quirks and all.

In the model used in this transducer, inflectional endings have the following properties:

- an upper-language form consisting of a string of grammatical tags
- one or more lower-language forms (including allomorphs and dialectal variants), possibly conditioned by specific phonological environments
- a category code
- the possibility to belong to more than one formative class

The inflection file was designed to take advantage of an idiom commonly used in conjunction with the Tcl programming language, alternately called “data as code”⁶ and “data is code”⁷. This idiom involves structuring data in such a way that it conforms to standard Tcl syntax, with each meaningful line beginning with a keyword. For each keyword, a procedure is defined which performs some operation using the rest of the line as input. In the case of the inflection file, the operation is to convert the data into an XML-based format (see Section 3.6 on pages 108-110). Once the appropriate procedures are loaded into the Tcl interpreter, the data can then be loaded into the interpreter as a Tcl program. The advantage of this idiom is that it leverages the Tcl parser, eliminating the need to write separate code to load and parse the data.

Tcl syntax is governed by twelve rules,⁸ facetiously referred to as the “dodekatalogue”. Of interest at present are the following points:

- a Tcl script is a string of commands; commands are separated by newlines (but see the point on curly braces below)
- a command consists of words, which are separated by whitespace; the first word is used to identify a procedure to evaluate the command; the remaining words are passed to that procedure as arguments
- everything inside a pair of curly braces is treated as a single word; because of this, any newlines that may occur within curly braces do not have the effect of separating commands
- if the first non-whitespace character in a command is a hash mark (#), the entire line is treated as a comment and ignored

3.2.3.1 Category Codes and Metadata

Instead of sections, the inflection file has the following top-level keywords: ‘Categories’, ‘AdditionalCategories’, ‘Continuations’, ‘AdditionalContinuations’, ‘LongDistanceDependencies’, ‘AdditionalLongDistanceDependencies’, and ‘Table’. With the exception of ‘Table’, each of these keywords takes a single argument, a paired list surrounded by curly braces, with each pair separated by a newline. Following is an explanation of each keyword and its argument.

The ‘Categories’ and ‘AdditionalCategories’ parts of the inflection file are reproduced in Figure 3.11 on the next page. Keyword ‘Categories’ is analogous to the Categories section in the postbase file; it associates category codes with formative classes. The argument consists of a paired list where each line

⁶See http://groups.google.com/group/comp.lang.tcl/browse_frm/thread/57b936b3d6bc34af/e63dd0c48a985f3c.

⁷See “data is code,” <http://wiki.tcl.tk/17869>.

⁸The rules are documented in the Tcl manual page ‘Tcl’, a copy of which can be found at <http://www.tcl.tk/man/tcl8.5/TclCmd/Tcl.htm>.

lists a category code followed by a space followed by the name of a formative class. Keyword ‘AdditionalCategories’ allows inflectional endings matching specific patterns to be included in additional formative classes. This mechanism was created to allow forms of demonstrative pronoun *igña* which begin with *irr-* to be inflected only with endings actually attested with that stem allomorph (see pages 74–75), and to allow positional noun stems to be inflected with possessive endings only (see page 29). The list in the argument to ‘AdditionalCategories’ is also a paired list; the first item in the pair is a Tcl regular expression to be matched against the grammatical tags associated with each inflectional ending defined in the file; any ending whose tags match the pattern is included in the specified formative class (the second item in the pair). In the listing below, the first regular expression matches all endings tagged as nominal which are followed by a grammatical person tag; this class is defined in order to facilitate the implementation of positional bases (see page 29). The second pattern matches all endings tagged as demonstrative pronouns (+Dem+Pro) in any case other than absolutive and which have a singular grammatical number tag; this category is necessary for the implementation of variant forms of the demonstrative pronoun *igña* which begin with *irr-* (see pages 74–75). The third pattern is similar in form and purpose to the previous pattern. It matches non-singular demonstrative pronoun endings; it exists to facilitate the variant forms of the demonstrative pronoun *manna* that begin with *mak(k)-* rather than *matk-* (see page 75). The remaining patterns match endings for the negative contemporative, contemporative 1, consequential, and conditional, respectively, and are used for postbases which are restricted to specific moods.

```

Categories {
  n NounInflection
  i IntransitiveInflection
  t TransitiveInflection
  da DemAdvInflection
  dp DemProInflection
}

AdditionalCategories {
  \\+N.*\\+[1-3]R?(?:Sg|Du|Pl) PossessiveNounInflection
  \\+Dem\\+Pro\\+(?:Rel|Loc|Via|Abl|Trm|Sim|Mod)\\+Sg DemProNonAbsInflection
  \\+Dem\\+Pro.*\\+(?:Du|Pl) DemProNonSingularInflection
  .*\\+ContNeg.* ContNegInflection
  .*\\+Cont1.* Cont1Inflection
  .*\\+Conseq.* ConseqInflection
  .*\\+Cond.* CondInflection
}

```

FIGURE 3.11: Category definitions in inflection data file

Continuations defined in the inflection file are given in Figure 3.12 on the next page. As with categories, there are two sets of continuations defined. Keyword ‘Continuations’ defines continuations on the basis of category codes; the format is analogous to that used in the argument to ‘Categories’. Keyword ‘AdditionalContinuations’ defines continuations based on grammatical tag patterns. Additional continuations facilitate post-inflectional derivation by allowing inflectional endings whose grammar tags match

specified patterns to have two continuations: the usual continuation to enclitics, reduced forms, or the end of the word, and an extra continuation to a class of derivational suffixes.

```
Continuations {
  n EncliticsOrEnd
  i EncliticsOrEnd
  t EncliticsOrEnd
  da EncliticsOrEnd
  dp EncliticsOrEnd
}

AdditionalContinuations {
  \\+Dem\\+Adv\\+Loc DemAdvLocPostInflection
  \\+Dem\\+Adv\\+Abl DemAdvAblPostInflection
  \\+Dem\\+Adv\\+Trm DemAdvTrmPostInflection
  \\+Dem\\+Adv\\+Via DemAdvViaPostInflection
  \\+Dem\\+Pro\\+Loc DemProLocPostInflection
  \\+Dem.* DemAnyPostInflection
  \\+N\\+Loc.* LocativeNounPostInflection
  \\+N\\+Trm.* TerminalisNounPostInflection
}
```

FIGURE 3.12: Continuations defined in inflection data file

Long-distance dependencies are also defined in two ways, as shown in Figure 3.13. Long-distance dependencies due to verb valence are handled defined using the keyword ‘LongDistanceDependencies’ using category codes and “require” flag diacritics (see Section 2.3.5 on pages 55–58). Long-distance dependencies due to number restrictions on nouns are defined with the keyword ‘AdditionalLongDistanceDependencies’ using grammatical tag patterns and unification flag diacritics. The patterns essentially associate each nominal inflectional ending of a specific grammatical number with a corresponding flag diacritic.

```
LongDistanceDependencies {
  i @R.VALENCE.INTR@
  t @R.VALENCE.TR@
}

AdditionalLongDistanceDependencies {
  \\+N.*\\+Sg @U.NUMBER.SG@
  \\+N.*\\+Du @U.NUMBER.DU@
  \\+N.*\\+Pl @U.NUMBER.PL@
}
```

FIGURE 3.13: Long distance dependencies defined in inflection data file

Flag diacritics on inflectional endings serve to filter out disallowed stem⁹-inflection combinations, such as the use of a transitive verb ending with an intransitive-only verb stem. The flag diacritics on verb inflections are “require” diacritics because all verb stems should be flagged for valence, and those flags will either agree with the flags on the inflectional endings (in which case concatenation of the stem and inflection is allowed in the transducer) or not (in which case the stem-inflection combination is filtered out of the transducer). In contrast to verbs, most noun stems can take any nominal inflection and so are not flagged. Because of this, noun inflection diacritics use unification: only stem-inflection combinations where the stem has a flag diacritic whose value is incompatible with the inflection’s flag diacritic are filtered; combinations containing stems without flag diacritics and stems with flag diacritics whose value does not conflict with the flag diacritic on the inflection are allowed. (See pages 70–71 for a discussion of flag diacritics attached to stems, and pages 77–78 for a discussion of flag diacritics attached to postbases.)

3.2.3.2 Inflection Table Syntax

After the category codes and metadata, the rest of the file consists of instances of the ‘Table’ keyword. Tables define two-dimensional inflectional paradigms whose upper-language representations are strings of grammar tags and whose lower-language representations are surface forms similar to those in the postbase section; they usually begin with a suffix attachment symbol (see pages 79–80), and there may be multiple allomorphs or dialectal variants for a single member of a paradigm. Transitive verb paradigms also have “holes” (slots in the table where no legal suffix exists); this is because coreference between the subject and object of a verb is expressed by using an intransitive inflectional suffix.

Tables are made of rows and columns. Specific grammatical tags can be associated with the table itself and with each row and column. The upper-language representation of an inflectional ending is generated by concatenating the corresponding grammar tags in this order: table tag(s) + row tag(s) + column tag(s). Lower-language representations are specified within the row.

The ‘Table’ keyword takes three arguments: a category code (which applies to each inflectional ending in the table), a string of table-level grammatical tags, and the table contents, enclosed in curly braces. The table contents are also evaluated as Tcl code. Two keywords are allowed inside the table contents: ‘Columns’ and ‘Row’. ‘Columns’ takes one argument: a whitespace-separated list of grammatical tag strings, one for each column in the table. ‘Row’ may be invoked in one of three ways:

- Row <rowTags> <rowContents>
- Row <rowTags> -prefixes <prefixList> <rowContents>
- Row <rowTags> -prefixset <prefixSet> <rowContents>

<rowTags> is a string of grammatical tags that apply to all inflectional endings in that row; <rowContents> is a list of surface forms of suffixes separated by one or more whitespace characters (possibly including newlines). Suffixes with allomorphs or dialectal variants are enclosed in curly braces and consist of whitespace-separated lists of conditioning patterns (see Section 3.3 on pages 96–104) and orthographic forms. Zero morphs, such as the absolutive singular ending, may be specified using an empty set of curly

⁹“Stem” in this paragraph includes stems derived from other stems by means of one or more postbases.

braces: `{}`. Holes in a paradigm are specified with the character ‘0’ (zero). Comments may be added in one of two forms, consistent with the Tcl commenting rules:¹⁰ comments on a line that contains nothing else but whitespace begin with a hash mark; comments at the end of a line containing non-whitespace begin with a semi-colon, optional whitespace, and a hash mark.

Figure 3.14 on the next page illustrates a table with the most basic type of row (one that does not specify `-prefixes` or `-prefixset`). This is extracted from the table specifying nominal inflection. While the table in the figure contains only the row corresponding to locative singular endings, the actual table in the inflection file contains 24 rows, one for each possible combination of case and number. The columns in this table are grammatical possessors. The first column represents unpossessed endings; accordingly, the grammar tag for that column is an empty string (specified with an empty pair of curly braces). The table has the tag `+N` in order to mark all nouns as such (see footnote 4 on page 70).

A few items in the table deserve additional explanation. The columns and row contents are broken into several lines, one for each grammatical person. This is done strictly to improve human readability; it has no impact on how the computer will interpret the table. The unpossessed locative singular ending begins with the rule trigger `OPTNSTEM`, which indicates that, when this suffix is attached to nouns whose absolutive singular form ends in `<n>`, the `<n>` may be left as is or converted back to its underlying form, `/ti/` (see Section 2.1.2.7 on pages 22–23); thus, for *apqun* ‘road’, both *apqunmi* and *apqutimi* are acceptable locative singular forms (MacLean 1986a:115). For each third person non-reflexive possessive ending, two variants are given, one which applies in all cases (preceded by condition code `?Always`), the other of which applies in cases where the preceding stem does not end in a vowel cluster (preceded by condition code `?notVthenV`). This is not (only) a case of allomorphy. The forms beginning with `-ŋ` were originally allomorphs of the forms beginning with `:a`, occurring only when the preceding stem ended in a vowel cluster. However, due to relatively recent innovation, the forms beginning with `-ŋ` have come to be accepted in all phonological environments. Forms beginning with `:a` are more conservative (MacLean 1986a:166; Kaplan 1981c:188–192).

The basic row mechanism works well for nominal inflection but leaves something to be desired for verbal inflection. With few exceptions, verbal inflectional endings in Iñupiaq may be divided into a mood marker and a suffix indicating the grammatical person and number of the subject and object (if any). For example, the indicative “present” intransitive mood marker is `-tu-` (`-ru-` after vowels). To this marker may be added person suffixes, such as `-ŋa` ‘I’, `-sî` ‘you (three or more)’, or `-k` ‘they (two)’. Different moods require different sets of person suffixes, and in several moods the mood marker changes depending on the grammatical person of the subject (most frequently, third-person non-reflexive subjects have a different mood marker than other subjects). The imperative mood has no obvious mood marker.

The most naïve way to deal with this within the table framework would be to treat each verbal inflectional suffix as a compound, entered in a row like this: `{?C +tuŋa ?V +ruŋa} {?C +tuguk ?V +ruguk} {?C +tugut ?V +rugut}` etc. This would work, but would be rather redundant. Alternatively, verb inflections could be separated into mood markers and person suffixes; this is the approach taken by Langgård

¹⁰Despite their surface appearance, comments inside the row contents are not actually treated by the Tcl parser as comments, because the row contents are never evaluated as a set of commands—they are simply an argument to the `Row` command. This command contains instructions to strip out comments before processing the row contents. Tcl comment syntax was used within rows to maintain consistency with comments elsewhere in the file, which are actual Tcl comments and treated as such by the Tcl interpreter.


```

Table n +N {
  Columns {
    {}
    +1Sg +1Du +1Pl
    +2Sg +2Du +2Pl
    +3Sg +3Du +3Pl
    +3RSg +3RDu +3RPl
  }
  Row +Loc+Sg {
    # row is split into sections for ease of reference
    OPTNSTEM/mi ;# no possessor
    -mnl -ptignl -ptinnl ;# 1st person possessor
    +ɲni -vsigñi -vsiññi ;# 2nd person possessor
    {?Always -ɲanl ?notVthenV :anl}
    {?Always -ɲaɲnl ?notVthenV :aɲnl}
    {?Always -ɲannl ?notVthenV :annl} ;# 3rd person possessor
    /miñl /mignl /miɲnl ;# 3rd person reflexive possessor
  }
}

```

FIGURE 3.14: Example inflectional table with no prefixes

and Trosterud. This would make the specification more compact, but would require additional formative classes and continuation classes. This approach is also complicated by the fact that, while MacLean's grammars divide verbal inflections into mood markers and person suffixes, there is no analysis across moods of which person endings go with which mood markers; that analysis would have to be done before inflectional suffixes could be entered, creating additional work.

The solution I have adopted is somewhat of a hybrid between treating verbal inflections as compounds and separating them into mood markers and person endings. For each row, it is possible to specify "prefixes" which will be attached to each string in the row contents. The prefixes reduce redundancy, like the "separate mood marker" approach (albeit to a lesser extent); but because the prefixes are associated with specific rows, they eliminate the need for separate formative classes for mood markers and person suffixes, like the "compound suffix" approach. Thus the prefix approach allows mood markers and person suffixes to be teased apart, yet contained within the same table; in this respect, it models the way verbal inflections are presented by MacLean (1986a, b, n.d.b) more closely than either other approach, facilitating data entry.

There are two ways to specify prefixes; both make it possible to specify variant forms. The simplest way, used in the majority of cases, is to use the keyword -prefixes followed by a paired list of conditioning patterns and orthographic forms, surrounded by curly braces. Two rows using this format are shown in Figure 3.15 on the next page; these were extracted from a larger table in the inflection file. The table contains intransitive verbal inflections and is tagged +V; the table's category code is 'i' (tables containing transitive inflections are also tagged +V but have category code 't'). The columns in this table correspond to grammatical subjects. Note that different verb moods correspond to different subjects; for example, the imperative mood applies only to second person subjects, while the optative mood is used only in first and third person; most dependent moods have a set of third person reflexive (sometimes called fourth person)

forms, while independent verb moods by definition do not. Each unique set of subjects requires a different set of columns and therefore a separate table.

```

Table i +V {
  Columns {
    +1Sg +1Du +1Pl
    +2Sg +2Du +2Pl
    +3Sg +3Du +3Pl
  }
  Row +Ind+Prs -prefixes {?C +tu ?mml +u ?Otherwise +ru} {
    ŋa guk gut
    tin {sik tik} sl
    q k t
  }
  Row +Simul2 -prefixes {?Always -ŋŋaq} {
    +ma -mnuk -pta
    {+kplt +kpln} -vsik -vsl
    -an -aŋnik -isa
  }
}

```

FIGURE 3.15: Example inflectional table using rows with the -prefixes option

The two rows in Figure 3.15 correspond to the indicative “present” and the simultaneitive II moods, respectively.¹¹ The indicative present mood marker has two main allomorphs, *-tu-* (which occurs following a consonant) and *-ru-* (which occurs following a vowel). However, the postbase *-mmî-* ‘also’ deletes the initial consonant of the indicative present mood marker. Because it does not have this effect on any other morpheme, the easiest way to deal with it is to treat the mood marker minus its initial consonant as another allomorph, one with a very specific conditioning environment. The allomorph *-tu-* is predictably associated with the condition code ?C, and the “allomorph” *-u-* is unsurprisingly associated with the code ?mml, but allomorph *-ru-* is not associated with the code ?V as one might expect. The reason is that ?V would incorrectly allow the sequence *-mmî-ru-*. Instead, the special condition code ?Otherwise is used. This code signals that the corresponding allomorph is used whenever all other conditions in the prefix set do not apply—in this case, whenever the preceding stem does not end with a consonant or with the morpheme *-mmî-*. The magic behind ?Otherwise is described on pages 100–101. The simultaneitive II mood marker is *-ŋŋaq-* in all cases; it is not subject to any allomorphy except as produced by synchronically productive rules. Accordingly, the special condition code ?Always is used.

Both rows contain cells surrounded with curly braces; in these cases, a surface form is defined for each combination of each prefix and each suffix. For example, in the case of the indicative present second

¹¹The indicative “present” is the default verb mood and generally expresses events and actions which are either ongoing or recently completed (MacLean 1986a:65). Over time, the Iñupiaq participial mood has been reinterpreted as an indicative past mood (the participial usage is retained as well) and the original indicative has acquired a present meaning.

The simultaneitive II mood is a dependent verb mood used for events and actions taking place at the same time as the event or action described by the main verb. The subject of a simultaneitive II verb is never coreferential with the subject of the main verb (MacLean n.d.b:chapter 23, page 1).

person dual, a total of six surface forms are associated with the lexical form +V+Ind+Prs+2Du: +tusik, +tutik, +usik, +utik, +rusik, and +rutik.

Contents of the simultaneous II row begin with suffix attachment symbols (see pages 79-80), while contents of the indicative present row do not. This is because no special morphophonological interaction occurs after the indicative present mood marker, whereas the simultaneous II mood marker ends with a ⟨q⟩ which is retained or deleted depending on the specific personal suffix. Suffix attachment symbols are available if necessary and add to the descriptive power of the prefix mechanism, but they need not be used in situations where simple concatenation of prefixes and suffixes is sufficient.

Most verbal inflections can be handled using -prefixes, but some verb moods involve a more complex interaction between mood markers and person suffixes. The conditional is such a mood.¹² If the subject is third person non-reflexive (that is, not coreferential with the subject of the main verb), the mood marker -kpa- is used; otherwise, the mood marker is -ka-, which becomes -ga- when preceded by a vowel, and -ġa- when the stem ends in ⟨q⟩ (the ⟨ġ⟩ takes the place of the ⟨q⟩). Rows invoking the -prefixset argument can handle such complexities.

Figure 3.16 shows a table implementing the conditional intransitive endings using -prefixset. The prefix set consists of a whitespace-separated paired list of keys and prefix lists. Each prefix list has the same format as the prefix lists used with the -prefixes option. Keys may be arbitrary characters, but should be non-alphabetic; the key 0 may not be used. The string default is also allowed as a key and has special meaning. Each suffix in the row contents may begin with a key from the prefix set, in which case the prefixes from the corresponding list are applied. If the suffix does not begin with a key, then the default prefixes are applied. (The -prefixes option is equivalent to specifying a prefix set whose only key is default.) In Figure 3.16, the default prefix list is applied to all person suffixes except for the third person non-reflexive suffixes, which are marked with the key of the other prefix list, 1.

```

Table i +V {
  Columns {
    +1Sg +1Du +1Pl
    +2Sg +2Du +2Pl
    +3Sg +3Du +3Pl
    +3RSg +3RDu +3RPl
  }
  Row +Cond -prefixset {
    default {?V +gu ?AnyQ -ġu ?Otherwise +ku}
    1 {?Always +kpa}
  } {
    ma mnuk pta
    {vlt vln} vsik vsl
    1n 1ġnik 1ta
    mI mik {mik miġ}
  }
}

```

FIGURE 3.16: Example inflectional table using rows with the -prefixset option

¹²The conditional mood is a dependent mood expressing concepts such as “when (in the future)” and “if” (MacLean 1986b:93).

One final detail about inflection table syntax that needs to be mentioned is the use of an asterisk either as the first character in a cell or as the first character in a prefix. In either case, the asterisk was intended to mark the corresponding formative or (in the case of a prefix) formatives as non-standard, that is, either innovative or archaic. With this information, it would be possible to parametrize the process of compiling the transducer to produce either an inclusive or an exclusive transducer; the former could be used for recognition, while the latter could be used for generation. In retrospect, limiting the parametrization to a single binary property was short-sighted, and the capability of specifying parameters to the process which builds the transducer has not been implemented. However, this feature, if properly developed, could be very beneficial, and warrants additional planning and development.

3.2.3.3 Accommodating Complex Paradigms Using Two-Dimensional Tables

As mentioned in Section 2.1.4.1 on page 27, nouns are inflected for case, number, and possessor, if any. Thus, the nominal paradigm has at least three axes (more if one divides the possessor axis into presence/absence of a possessor, grammatical person of the possessor, and grammatical number of the possessor). Verbs are inflected for mood, subject, and object, if any. Demonstrative adverbs are inflected only for case, while demonstrative pronouns are inflected for case and number. Since only one of the four paradigms is actually two-dimensional, one may wonder why the inflectional model used here is inherently restricted to two dimensions, rather than supporting a variable number of dimensions. The choice to use two-dimensional tables was based on the fact that most inflectional endings in MacLean's books are presented in such tables. In hindsight, it may have been a good idea to consider a framework where the number of dimensions in a table could fluctuate, but the two-dimensional model has proven quite adequate. Four key features add flexibility to the model:

- tables are not exclusive—it's permissible to define more than one table with the same category code, grammatical tag string, and/or column definition
- with the help of newlines, spaces, and comments, row data can be made more readable (for example, all 1st person endings can be grouped together on their own line)
- empty strings are allowed both as tag strings and as surface forms
- it's possible for a table to contain "holes"—positions in the paradigm where no suffix exists

The rest of this section explores the implementation of each set of inflectional endings.

3.2.3.3.1 NOMINAL INFLECTION All nominal inflectional suffixes are contained in a single table. The columns of the table are possessors; the first column is for unpossessed nouns (the grammatical tag for that column is an empty string). The rows in the table correspond to the Cartesian product of possible values for case and number; in other words, one row contains absolute singular endings, the next absolute dual endings, then absolute plural, then relative singular, and so on.

3.2.3.3.2 INTRANSITIVE VERBAL INFLECTION Intransitive inflection is divided into eight separate tables. For most of these tables, the columns correspond to subjects, and the rows correspond to verb moods. The largest table contains the following moods: indicative present and past, interrogative, contemporative I realized and unrealized, negative contemporative, contemporative II realized and unrealized, simultaneitive

II, and *kiisaimmaa*. Each of these moods applies to the set of first, second, and non-reflexive third person subjects. Rows corresponding to the interrogative, negative contemporative, simultaneitive II, and *kiisaimmaa* moods have one grammatical tag each, while rows corresponding to the indicative, contemporative I, and contemporative II moods each have two tags (e.g., +Ind+Prs for indicative present; +Cont2+Real for contemporative II realized).

The next table covers the simultaneitive I and III moods. The subject of a simultaneitive I or III verb must be coreferential with the subject of the main verb, so the applicable subjects are first, second, and reflexive third persons.

Optative and imperative moods are each given their own table. The optative table applies to first and non-reflexive third person subjects, while the imperative table applies to second person subjects.

Consequential and conditional moods share a table. Possible subjects for these moods include first, second, non-reflexive third and reflexive third persons.

The remaining three tables implement verbal nouns (participials and gerunds). Controversially, these tables are tagged +V just like all the other verbal inflectional tables, despite the fact that verbs so inflected are syntactically nouns. This decision was made on the grounds that these inflectional endings apply only to verb stems; in any case, the fact that they are marked as gerunds or participials should provide sufficient notice of their nominal status.

The participial is split into two tables, one covering first and second person forms, the other covering third person forms. The form of first and second person participials remains the same whether they are objects of a transitive verb or subjects of a transitive or intransitive verb; this suggests that it is inappropriate to tag these forms with case markers. However, these participials may be optionally marked as vocative by lengthening the vowel of the final syllable. To account for these facts, the columns in the table of participials are first and second person subjects with and without vocative marking (the actual column declaration is {+1Sg +1Du +1Pl +1Sg+Voc +1Du+Voc +1Pl+Voc +2Sg +2Du +2Pl +2Sg+Voc +2Du+Voc +2Pl+Voc}). The table contains a single row, tagged +Part.

Third person participials occur in all noun cases but have no vocative form. For ease of input, the third person participial endings are listed in a single row, tagged +Part; the columns in the table are the Cartesian product of the sets {+3Sg, +3Du, +3Pl} and {+Abs, +Rel, +Mod, +Abl, +Trm, +Loc, +Via, +Sim}. Of course, it would have been possible to arrange the table with one row for each third person number, in which case only eight columns would have been required rather than 24. I declined to use this format because it would have required me to enter the suffixes in a different order than they appear in MacLean's work.

The gerund table is very simple; it contains one column, tagged with an empty string, and one row, tagged +Gerund. The table format is clearly not ideal for individual suffixes such as the gerund (which is simply the relative case marker attached to a verb stem), but it can be made to accommodate them nonetheless.

3.2.3.3.3 TRANSITIVE VERBAL INFLECTION With transitive verbs, each verb mood is given its own table. All mood-related tags are included with the verb tag in the table tags; for example, the tag string associated with the table for the indicative present mood is +V+Ind+Prs. The columns in each table correspond to direct objects. The rows correspond to grammatical subjects, except in the cases of the contemporative I and

II and negative contemporative moods, which are not inflected for subject when transitive. The contemporative I and II tables have a row each for realized and unrealized aspect; the negative contemporative mood has only one row, whose tag string is empty.

As mentioned on page 87, because coreference of subject and object is expressed by using intransitive endings, transitive paradigm tables contain “holes”—cells for which no suffix exists. In the format used in the inflection file, these cells are marked with zeros (see on page 88). Figure 3.17 contains the first person singular row from the transitive interrogative mood table. Because the suffixes in this row have a first-person subject, they cannot also have a first-person object, so these columns contain zeros.

```
Table t +V+Int {
  Columns {
    +1SgO +1DuO +1PIO
    +2SgO +2DuO +2PIO
    +3SgO +3DuO +3PIO
  }
  Row +1Sg -prefixes {?V +vI ?C +pI} {
    0 0 0
    giñ sik sl
    gu gik {gl gIt}
  }
}
```

FIGURE 3.17: Example inflectional table with “holes” (cells for which no suffix exists), marked by zeros

3.2.3.3.4 DEMONSTRATIVE INFLECTION As discussed on pages 74–75, demonstratives may be inflected either as pronouns or as adverbs, but because non-linguists are unlikely to be familiar with bare demonstrative stems, the stem file lists demonstratives separately as adverbs (in “interjectional” form) and as pronouns (in absolutive singular form). Special sets of morphographemic rules transform these demonstratives into stems to which other inflectional suffixes can be added. These rules are sensitive to the triggers DASTEM (for demonstrative adverbs) and DPSTEM (for demonstrative pronouns); an additional trigger, DELETEEXTRAVOWEL is used with the demonstrative adverb locative case ending *-uuna* to prevent three-vowel clusters. These triggers are prefixed to the non-default inflectional suffixes; the default suffixes are left empty, since the demonstratives are listed in the stem file in fully-inflected forms. In the demonstrative adverb table, the columns are cases (the first column being associated with an empty string, corresponding to the interjectional form), and there is only one row, with no tags. In the demonstrative pronoun table, the columns are grammatical numbers, and the rows are cases. The demonstrative pronoun table takes advantage of the *-prefixes* option to apply the rule trigger DPSTEM to non-absolutive suffixes (this technique cannot be used with demonstrative adverbs because the row contains the interjectional form, which should not be reduced to a stem). The demonstrative tables are reproduced in Figure 3.18 on the next page.

```

Table da +Dem+Adv {
  Columns {{} +Loc +Via +Abl +Trm}
  Row {} {
    {} DASTEMani DASTEMDELETEEXTRAVOWELuuna DASTEMarjga DASTEMurja
  }
}

Table dp +Dem+Pro {
  Columns {+Sg +Du +Pl}
  Row +Abs {{} DPSTEM-kuak DPSTEM-kua}
  Row +Rel -prefixes {?Always DPSTEM} {uma -kuak -kua}
  Row +Loc -prefixes {?Always DPSTEM} {umanl -kuknagnl -kunanl}
  Row +Via -prefixes {?Always DPSTEM} {umuuna -kuknuuna -kunuuna}
  Row +Abl -prefixes {?Always DPSTEM} {umarjga -kuknarjga -kunarjga}
  Row +Trm -prefixes {?Always DPSTEM} {umurja -kuknurja -kunarja}
  Row +Sim -prefixes {?Always DPSTEM} {umatun -kuknaktun -kunatitun}
  Row +Mod -prefixes {?Always DPSTEM} {umirja -kuknirja -kunirja}
}

```

FIGURE 3.18: Inflectional tables implementing demonstrative inflection

3.2.3.4 Sources and Data Entry Methodology

All inflectional endings came from MacLean (1986*a, b*, n.d.*b*). Inflectional endings from these sources are presented as tables; the data in the tables were rearranged as necessary to ensure that grammar tags were concatenated in the appropriate order; see Section 3.2.3.3 for details on how specific paradigms are specified in the inflection file.

3.2.4 Enclitics

Although enclitics are syntactically independent words, they are phonologically and orthographically bound to the words that precede them. Also, certain enclitics appear to attach only to words of a particular grammatical category, inflected with specific endings. In these respects, enclitics are similar to postbases, and for this reason, the same file format that is used for postbases is used for enclitics (see pages 77-80 for an explanation of the syntax). The enclitic file is much simpler, however; enclitics are not subject to long-distance dependencies, and the morphotactics of enclitics are certainly less complex than those of postbases (the present implementation probably oversimplifies here, allowing enclitics to continue to the class of all reduced forms and enclitics that are not restricted to specific grammatical categories).

Figure 3.19 on the following page gives the category and continuation definitions from the enclitic file. Most enclitics fall under attachment category code ‘e’ indicating that they are not restricted to specific grammatical categories or inflections. Enclitic *-tchiaq* ‘by oneself’ (Edna MacLean, personal communication, 2 July 2009) is categorized as applying only to forms of the pronoun *kisi* ‘X alone,’ since I am not aware of other instances of this enclitic. Homophonic enclitic *-tchiaq* ‘on the ____ side of’ (Edna MacLean, personal communication, 2 July 2009) is listed with appropriate case endings in categories ‘daLoc’, ‘daTrm’, and ‘daAbl’ (for example, the entry associated with the demonstrative locative is *tchiaq>+Loc,-tchianl*

daLoc ee). This *-tchiaq* might be better categorized as a post-inflectional suffix; see footnote 17 on page 30. Suffix *-gruiññaq* is also listed as an enclitic in the ‘daTrm’ and ‘daAbl’ categories, because it appears in the development data of the test corpus (see Section 4.1 page 146) in the words *tamaannaqgruiññaq* ‘out of nowhere/from nowhere’ and *taimuñaggruiññaq* ‘for a long period of time’ (translations provided by Ronald Brower, 6 April 2011). Enclitic *-tuq* ‘I hope’ occurs only in sentences with verbs in the optative mood, but is listed with category code ‘e’ rather than a more restrictive code because, like many enclitics, it generally attaches to the first word in the sentence, whether or not that word is a verb in the optative mood (MacLean [1986b:32-33]; see (67)).

```
SECTION Categories
e Enclitics
kisi KisiSuffixes
daLoc DemAdvLocPostInflection
daTrm DemAdvTrmPostInflection
daAbl DemAdvAblPostInflection

SECTION Continuations
ee EncliticsOrEnd
```

FIGURE 3.19: Category and continuation definitions in enclitic data file

Ex. (67) *Uvlupaktuq Aaṅatkut aḡvaglit*
uvlupak=tuq Aaṅa-tkut aḡvak-lît
 today=I.hope Aaṅa-party.of.abs.pl catch.a.whale-OPt.3Pl
 ‘I hope Aaṅa and his crew get a whale today’ (MacLean 1986b:33)

The sources for the enclitics listed in this file are MacLean (1981, n.d.a).

3.3 Definitions of Allomorph Conditioning Environments

As discussed in 2.1.2.8 on pages 23–25, a large number of Iñupiaq suffixes exhibit allomorphy. Langgârd and Trosterud’s strategy for handling this allomorphy is to group formatives that require the same phonological environment into the same class, and to establish continuation classes so that formatives which fulfill particular environments continue to classes of formatives which require those environments. For example, verbal formatives that end in a vowel continue to a class of deverbal suffixes that are allowed to follow vowels. The challenge that this approach creates is described in Section 2.4.3.3 on pages 64–66; briefly, the substantial number of conditioning environments requires a complex maze of formative classes and continuations. In the Iñupiaq transducer, due to the format used for specifying the lexicon, the problem with this approach is not how to construct such a maze by hand, but how to instruct a computer to do so.

My initial approach to this problem was to create a lexc file similar to the one used by Langgârd and

Trosterud (n.d.; see Section 2.4.3 on pages 61–66). First, for each conditioning environment to which an allomorph in the lexicon is sensitive, I created a Tcl regular expression that would match strings containing that environment. I then used the following algorithm to generate the lexc file:

1. group the allomorphs of each postbase and inflectional suffix by the grammatical category they attach to and the conditioning environment to which they are sensitive; a side-effect of this grouping procedure is to identify the conditioning environments that apply to each grammatical category
2. for each stem and postbase, apply the set of regular expressions that correspond to the conditioning environments of the grammatical category to which the stem or postbase belongs (in the case of postbases, this is the category the postbase derives); remember which set of conditions each stem or postbase satisfies
3. identify the set of unique sets of conditions satisfied by stems and postbases of each grammatical category (for example, there may be several nominal formatives which provide the environments “consonant-final”, “back-consonant-final”, “q-final”, and “‘weak-q’-final”; these four conditions would then constitute one unique set of conditions for the noun category)
4. create a formative class for each conditioning environment of each grammatical category (as identified in step 1), to contain the allomorphs of postbases and inflectional endings that require that environment
5. for each unique set of conditions identified in step 3, create a formative class of epsilon continuations (see page 68) to each of the relevant classes created in step 4
6. populate a class of stems as well as each class defined in step 4 such that each stem and postbase allomorph continues to the appropriate class from step 5; inflectional suffix allomorphs will continue to the class of enclitics or to the end of word marker

This approach was functional but inelegant, not in the least because it was as difficult to program as it is to describe, or because it took a long time to run. The biggest drawback, however, was that by generating, on the fly, an unknown number of formative classes, it became very difficult to create a stem guesser (see Section 2.3.7 on pages 59–60). Generally, one creates a stem guesser for each open class of stems; a stem guesser belonging to a particular class should continue to any classes containing suffixes that can be attached to stems of the class to which the stem guesser belongs. When that set of classes is generated automatically, it becomes more difficult to continue to all of them. More problematically, a stem guesser designed to generate all phonotactically possible stems of a particular grammatical category will violate the phonological constraints of each phonologically-defined continuation class. To solve that problem, separate stem guessers would have to be created to satisfy each condition that pertains to the grammatical class of the guesser, and each of these would need to continue to the appropriate continuation class. In order to create these separate guessers, one would either need to know which conditions apply to which grammatical category, which would defeat some of the point of generating the file automatically, or else one would need to generate the guessers along with the formative classes to which they continue, which would be possible but non-trivial.

An alternative to creating a lexicon that respects the phonological constraints of all allomorphs is to create an overgenerating lexicon that respects none of these constraints, and then to filter out generated words which contain allomorphs in disallowed environments (Bills et al. 2010:24). The filtering mechanism

is a set of regular expressions, written in xfst rather than in Tcl, which apply to entire words rather than individual formatives. For this strategy to work, allomorphs are marked with rule triggers to which the filtering expressions are sensitive; the filters disallow any strings which contain the rule trigger but not the appropriate preceding environment. After performing this filtering, the filter rules map the rule triggers to empty strings, in effect deleting the triggers from the surface forms of the words (separate rules clean up the upper-language representation of the lexicon so that none of the triggers are visible to the end-user in either lexical forms or surface forms).

Because the filters apply to full words, they filter out any guessed stems which violate the phonological constraints of allomorphs; thus, only one set of guessing rules is required per open stem class (in the case of Iñupiaq, nouns and verbs). This approach also makes it completely unnecessary to take phonology into account when establishing formative classes in the lexicon, allowing the linguist to focus on purely morphotactic considerations.

The file `data/patterns.txt` contains definitions for the allomorph conditioning environment filters used in the Iñupiaq transducer. The remainder of this section discusses the logic used by these filters and the syntax used to define them.

3.3.1 Logic

The rules that check conditions are based on logic described in Beesley and Karttunen (2003:252-253) and Kaplan and Kay (1994:345). Given some pattern P and an associated rule trigger T , a filter rule needs to disallow strings that contain T when preceded by anything that does not match P . We also need to make allowance for non-alphabetic symbols that may be present in the string, such as other rule triggers or flag diacritics. The way the rule is written depends in part on whether the pattern needs to take into consideration the left edge of the string to be matched (for example, the left edge is not needed to determine whether a stem is vowel-final, but it is needed to determine whether a stem is two syllables long). If, as in the majority of cases, the left edge is unimportant, the regular expression will follow the model given in Figure 3.20. In the discussion that follows, I will build up this expression one piece at a time, explaining the relevance of each part.

[[Pattern / SymbolsToIgnore] Trigger | \Trigger]*

FIGURE 3.20: Regular expression model for filters not sensitive to the left edge of a word

As previously stated, a filter should disallow strings that contain a trigger T preceded by a stem that does not match a pattern P . If, however, the string does not contain T , then the filter is irrelevant for that string and should let the string through (or in other words, the filter should recognize any string that does not contain T). Because finite-state automata cannot match the absence of something, we must construct our regular expression in terms of the presence of something else: symbols which are not the rule trigger in question. The term complement operator, which in xfst is a backslash (Beesley and Karttunen 2003:47), yields a language which is the set of all single symbols which are not in the language denoted by its

operand. Thus, $\setminus\text{Trigger}$ is the set of all symbols that are not the rule trigger defined by Trigger . We would like to be able to define the language of all strings which consist entirely of symbols that are not the rule trigger corresponding to our filter; the Kleene star iterator (*) will allow us to do this. Thus, our expression for strings that do not contain the rule trigger Trigger is $[\setminus\text{Trigger}]^*$.

We also want to recognize strings that do contain the rule trigger as well as the appropriate preceding environment. With very few exceptions, the preceding environment will be defined strictly in terms of letters of the Iñupiaq alphabet (for example, sequences of the form Vowel-Consonant-Vowel, which I denote with the expression $[\text{V C V}]$). However, words in the lexicon also contain non-alphabetic symbols such as flag diacritics, rule triggers, or morpheme boundaries; these should not be taken into account when determining whether the string follows the pattern associated with the filter. This can be accomplished with the ignore operator, indicated in xfst with a forward slash (Beesley and Karttunen 2003:48). The expression $[[\text{V C V}] / \text{SymbolsToIgnore}]$ creates a language where zero or more strings defined in SymbolsToIgnore may occur at the beginning or end or anywhere in the middle of the language defined by $[\text{V C V}]$, in effect ignoring any instances of SymbolsToIgnore within the pattern of interest. The effect of the ignore operator on an automaton is illustrated below; Figure 3.21 shows the basic pattern, which is modified in Figure 3.22 to ignore SymbolsToIgnore . The expression which recognizes the pattern of interest (here represented as Pattern), followed by the rule trigger (represented as Trigger), ignoring SymbolsToIgnore within the pattern, is $[[\text{Pattern} / \text{SymbolsToIgnore}] \text{Trigger}]$.

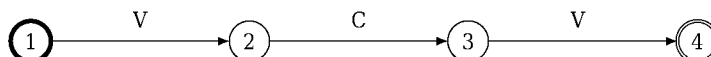


FIGURE 3.21: Automaton without ignore operator, illustrating the expression $[\text{V C V}]$

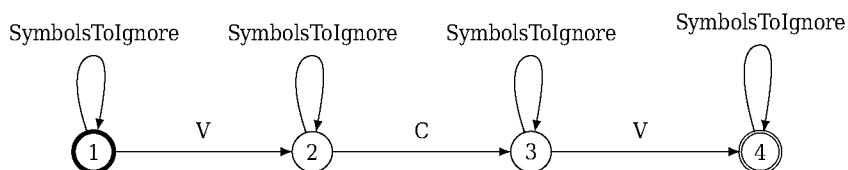


FIGURE 3.22: Automaton with ignore operator, illustrating the expression $[[\text{V C V}] / \text{SymbolsToIgnore}]$

We now have an expression that can match entire strings not containing a particular rule trigger, as well as an expression that can match the rule trigger and preceding pattern. This second expression is incomplete, since we must match an entire string and not just the substring containing the pattern and the trigger. There is a temptation to complete the expression by allowing any symbols on either side of the

pattern-and-trigger expression: `[?* [Pattern / SymbolsToIgnore] Trigger ?*]`.¹³ However, this will incorrectly allow strings in which the rule trigger appears twice, but only once following a substring that matches the pattern in question. To prevent this, we may be tempted to bracket the pattern-and-trigger expression with zero or more instances of symbols which are not the rule trigger: `[[\Trigger]* [Pattern / SymbolsToIgnore] Trigger [\Trigger]*]`. Unfortunately, this pattern will incorrectly disallow all strings containing more than one instance of the rule trigger. The correct solution is to allow strings built up of any combination of the following two types of substrings:

- substrings matching the relevant pattern (ignoring any intervening non-letter symbols) followed immediately by the rule trigger
- single-symbol substrings which are not the rule trigger

This approach unifies the pattern for strings not containing the rule trigger and strings containing the rule trigger and the appropriate conditioning environment. The rule trigger may appear in the string any number of times (including zero), but the preceding environment must match the appropriate pattern each time. The final expression, repeated from Figure 3.20, is `[[Pattern / SymbolsToIgnore] Trigger | \Trigger]*` (the pipe symbol (`|`) is the union operator).

If a formative has allomorphs whose conditioning environments are mutually exclusive, it is possible to designate one conditioning environment as the environment that applies when all others do not. This is done using the special condition code `?Otherwise`, and can be useful when a conditioning environment would be unusually complex to specify. `?Otherwise` was used with the indicative present intransitive endings shown in Figure 3.15 on page 90. Three mood marker allomorphs are defined for this paradigm: `-tu-`, which attaches to consonant-final stems; `-u-`, which attaches to stems ending in the postbase `-mmî-` ‘also’; and `-ru-`, which attaches to vowel-final stems except those ending in the postbase `-mmî-`. The regular expression for stems ending in vowels but not ending in `-mmî-` would be very difficult to write by hand, but it can be built automatically from expressions representing the “opposite” of the expressions matching consonant-final and `-mmî-` final stems.

Expressions “opposite” of the one in Figure 3.20 are built on the model shown in Figure 3.23. It’s important to note that even though this expression may compute the opposite language of the consonant-final condition, `Trigger` here refers to the rule trigger for the `?Otherwise` condition (that is, vowel-final stems not ending in `-mmî-`), not the consonant-final condition. The goal in calculating an expression for the `?Otherwise` condition is to disallow strings containing the trigger for this condition when preceded by a substring that matches one of the other allomorphs’ conditions. The regular expression defining a filter for the `?Otherwise` condition is created by intersecting the “opposite” expressions for all of the other conditions that pertain to the formative in question. In the case of the indicative present intransitive mood marker, it is the intersection of the opposite expression for consonant-final stems and the opposite expression for stems ending in `-mmî-`. This expression will match any string that does not contain the rule trigger for vowel-final stems not ending in `-mmî-`, as well as any string in which this trigger is preceded by a substring that is neither consonant-final nor `-mmî-` final.

The dollar sign is the xfst containment operator (Beesley and Karttunen 2003:48). The expression `[$[[Pattern / SymbolsToIgnore] Trigger]]` describes the set of all possible strings that contain the rule trigger

¹³The shorthand notation `[$[[Pattern / SymbolsToIgnore] Trigger]]` denotes exactly the same language.

~[\$[[Pattern / SymbolsToIgnore] Trigger]]

FIGURE 3.23: Regular expression for the “opposite” of the language expressed in Figure 3.20

preceded by a substring matching the pattern (possibly with one or more intervening non-letter symbols). The tilde is the xfst language complement operator (Beesley and Karttunen 2003:47); it computes the set of all strings not in the language of its operand. In this case, it computes the set of all strings that do not contain a substring matching [[Pattern / SymbolsToIgnore] Trigger].

There exists at least one allomorph which attaches only to stems containing a particular number of syllables: the imperative intransitive singular suffix *-iñ* that triggers gemination (see 24) attaches only to stems of the form (C)VCV. The filter for this condition must take into account every symbol from the rule trigger to the left edge of the string, and for obvious reasons, this particular rule trigger can occur no more than once in any given word; for these reasons, a different expression model is required than the one used for most conditions. The expression for this condition is [[[C] V C V] / SymbolsToIgnore] Trigger \Trigger* | \Trigger*]. It will match strings that follow one of two patterns:

- an optional consonant (denoted by parentheses) followed by a vowel, a consonant, and a vowel (with optional intervening non-letter symbols), followed by the appropriate rule trigger, followed by zero or more symbols which are not the rule trigger
- zero or more symbols, none of which is the rule trigger corresponding to this rule

The opposite pattern of this one is ~[[[(C) V C V] / SymbolsToIgnore] Trigger], which matches any string not containing an optional consonant (denoted by parentheses) followed by a vowel, a consonant, and a vowel (with optional intervening non-letter symbols), followed by the appropriate rule trigger (again, note that this refers to the trigger for the ?Otherwise condition, not the (C)VCV rule trigger), followed by zero or more symbols which are not the rule trigger.

3.3.2 Syntax

The file consists of three sections, XFST, Tcl, and Patterns, each of which begins with the keyword SECTION followed by a space and the name of the section. SECTION XFST is intended for definitions of character sets and patterns, written in xfst. These are not strictly necessary, but can be quite helpful for two reasons. First, they allow regular expressions, which can sometimes be hard to read, to be associated with meaningful names; these names can be used in subsequent expressions in place of the code they represent, making those expressions easier to read. Second, some patterns are used by more than one filter expression, and defining these patterns eliminates some redundancy.

The contents of SECTION XFST are given in Figure 3.24 on the following page. The first definition, SymbolsToIgnore, defines a language consisting of all single symbols other than those which represent graphemes in the Iñupiaq transducer (the definition of Graphemes, as well as definitions for Consonant and Vowel, are found in the file data/ipk_xfst.txt; see Section 3.8.2.2 on page 117). This definition is used in all of the filter expressions. Other definitions include StemFinalC, defining stem-final consonants (/k/, strong or weak /q/, and /t/ [palatalized or otherwise]); KQ, defining back consonants (/k/ and strong or weak /q/);

Strong, defining strong consonants (see page 6); 3Syll, which recognizes stems three or more syllables long (defined as a vowel, followed by more than one consecutive occurrence of the following pattern: one to three non-vowels followed by one or two vowels); CVCV, defining a pattern for stems consisting of an optional initial consonant, a vowel, another consonant, and another vowel (the pattern limits the first consonant to the set of attested Iñupiaq stem-initial consonants; see pages 25–26); and KQnotVV, defining a pattern for stems ending in a back consonant which is not preceded by a two-vowel cluster.

SECTION XFST

```
define SymbolsToIgnore \Graphemes ;
define StemFinalC [k | AnyQ | t ];
define KQ [k | AnyQ ];
define Strong [k | q];
define 3Syll [ \Vowel [ [\Vowel]^1,3 Vowel^1,2 ]^>1 ];
define CVCV [ ([p | m | t | n | s | y | k | q]) Vowel Consonant Vowel ];
define KQnotVV [ \Vowel Vowel KQ ];
```

FIGURE 3.24: Contents of the xfst section of the filter definition file

SECTION Tcl contains procedures defined in the Tcl programming language (see page 67 and page 84). These procedures generate xfst code and may be called from SECTION Patterns. Most of the definitions in SECTION Patterns will follow a single model, as discussed in Section 3.3.1 on pages 98–101, and the purpose of SECTION Tcl is to make it possible to define code that will generate the redundant parts of the pattern definitions automatically so that they need not be specified by hand each time that model is used. The contents of SECTION Tcl are given in Figure 3.25 on the following page. Procedure Positive creates an xfst expression applying the model shown in Figure 3.20 on page 98 to the pattern given in its argument. Procedure Negative creates an xfst expression applying the opposite model, shown in Figure 3.23 on the preceding page, to the pattern given in its argument. Procedure PositiveNoIgnoring does no ignoring, instead simply matching a string consisting of substrings not containing the rule trigger and/or substrings containing the rule trigger immediately preceded by the specified pattern. The corresponding procedure NegativeNoIgnoring similarly does no ignoring, matching strings for which no substring consists of the specified pattern followed by the rule trigger. These “no ignoring” procedures are used when the filter applies to a specific morpheme, such as *-mmî* ‘also’ which causes intransitive indicative present endings to lose their initial consonant. For such filters, it is desirable to ignore special symbols before and after the morpheme, but not inside it. For this reason, the task of specifying where and what to ignore is left to the person specifying the pattern. The implementation of the *-mmî* filter (among others) is given in Figure 3.26 on page 104.

The actual conditioning environment pattern definitions are contained in SECTION Patterns. A pattern definition contains three required elements: the pattern name (which by convention begins with a question mark), an xfst expression defining the pattern, and an xfst expression defining the opposite of the pattern (this is used to compute ?Otherwise condition patterns; see pages 100–101). The required elements may optionally be followed by a comment, beginning with an exclamation point. Blank lines and lines consisting

```

SECTION Tcl

proc Positive {pattern} {
    return [list [string map [list %p $pattern] {[[%p / SymbolsToIgnore ] %s | \%s}*]]]
}

proc Negative {pattern} {
    return [list [string map [list %p $pattern] {~[ [%p / SymbolsToIgnore ] %s ]}]]
}

proc PositiveNoIgnoring {pattern} {
    return [list [string map [list %p $pattern] {[[%p] %s | \%s}*]]]
}

proc NegativeNoIgnoring {pattern} {
    return [list [string map [list %p $pattern] {~[ [%p] %s ]}]]
}

```

FIGURE 3.25: Contents of the Tcl section of the filter definition file

solely of comments are also permitted in SECTION Patterns.

Procedures defined in SECTION Tcl may be used to generate patterns. These procedures are invoked by surrounding the procedure and its argument in square brackets; if the argument contains spaces, it must be enclosed in curly braces. It is also possible to specify a pattern in xfst directly; in that case, the pattern should be surrounded by curly braces. The pattern should include the string %s in place of the pattern name (which is a rule trigger); each instance of %s will be replaced with the pattern name when the pattern definition file is processed. The Tcl procedures Positive and Negative insert %s automatically into the expressions they generate.

Pattern expressions may also refer to variables defined in SECTION XFST.

Example entries from SECTION Patterns are given in Figure 3.26 on the following page. ?C matches stems with stem-final consonants; the pattern is defined using the Tcl procedures Positive and Negative, to which is passed the xfst variable StemFinalC which describes all legal stem-final consonants. ?VkAnyQ is similar to ?C except that the expression passed to Positive and Negative is slightly more complex; it is the disjunction of the xfst variables Vowel and KQ (in other words, this pattern will match stems whose final segment satisfies either Vowel or KQ). ?VthenVthenC, which matches stems ending in two vowels and a consonant, and ?notVthenVthenC, which matches stems not ending in two vowels and a consonant, are patterns in direct opposition to each other,¹⁴ and their definitions are identical except that for ?notVthenVthenC the calls to Positive and Negative are swapped. ?2SyllablesV matches stems of the form (C)VCV. As discussed on page 101, because this pattern matches a whole stem rather than just the end of the stem, the usual model is not appropriate; consequently, the pattern is defined directly in xfst rather than being generated by Tcl procedures. ?mml matches a stem ending in the suffix *-mmî-* ‘also’. It is defined with the help of the “no

¹⁴One may wonder why it was necessary to define ?notVthenVthenC, rather than use the ?Otherwise mechanism. The reason is that the postbase *-aluk* is listed in MacLean (n.d.a) with three allomorphs which are not entirely mutually exclusive: *-aluk* attaches to stems not ending in VVC; *-aaluk* attaches to stems ending in VVC; and *-gaaluk* attaches to stems whose absolutive singular form ends in {n} (this suffix deletes the {n}). Because stems whose absolutive singular form ends in {n} may be suffixed either by *-aluk* or by *-gaaluk*, it is impossible to use ?Otherwise in this case.

ignoring” procedures because the pattern contains a morpheme boundary, which is problematic for the “ignoring” procedures, and because there should not be any intervening non-alphabetic symbols within the string `m m i`. Instead, the ignoring of non-alphabetic symbols is explicitly specified in the pattern at the points where it is appropriate—immediately before and after the letters that make up the morpheme. To ensure that irrelevant symbols are only ignored at these points, the Kleene star (*) is used rather than the ignoring operator (/).

SECTION Patterns

```
?C [Positive StemFinalC] [Positive StemFinalC]
    ! matches any legal stem-final consonant (excluding n)

?VkAnyQ [Positive {[Vowel | KQ]}] [Negative {[Vowel | KQ]}]
    ! matches any stem-final vowel (including n) or stem-final k or q (weak or strong)

?vthenVthenC [Positive {[Vowel]^2 StemFinalC}] [Negative {[Vowel]^2 StemFinalC}]
    ! matches stems ending in two vowels and a consonant

?notVthenVthenC [Negative {[Vowel]^2 StemFinalC}] [Positive {[Vowel]^2 StemFinalC}]
    ! matches stems ending in two vowels and a consonant

?2SyllablesV {[CVCV / SymbolsToIgnore] %s \%s* | \%s*} {~[ [CVCV / SymbolsToIgnore] %s]}
    ! matches stems of the form (C)VCV-

?mml [PositiveNoIgnoring {%> SymbolsToIgnore* m m i SymbolsToIgnore* }]
    [NegativeNoIgnoring {%> SymbolsToIgnore* m m i SymbolsToIgnore* }]
    ! matches the postbase -mml-, after which an intransitive indicative present ending loses its initial consonant
```

FIGURE 3.26: Example entries from the Patterns section of the filter definition file

3.4 Formative Class Declarations

As discussed in Section 3.1, files in `lexc` are organized in terms of formative classes (called *lexicons* in `lexc` parlance); a class is named, then its members are enumerated (see Section 2.3.2 on pages 51–53). In that framework, epsilon continuations (see page 68) are created as class members which are empty strings.

In the `Iñupiaq` transducer, the lexicon is fundamentally organized in terms of stems, postbases, inflectional suffixes, and enclitics rather than in terms of formative classes (for rationale, see pages 58–59 and pages 69–70), so another mechanism is needed to declare formative classes and epsilon continuations.

Although formatives in our lexicon are not organized in terms of formative classes, each formative definition includes a reference to the class to which it belongs. One possible approach to declaring formative classes would be simply to create any classes that are referred to. This is similar to the way variables are created in dynamic languages such as Python and Tcl. The danger with this approach is that typographic errors would go undetected; if a class name were spelled correctly in one place but incorrectly

elsewhere, two distinct classes would be created, and this fact would not be brought to anyone's attention. Alternatively, classes could be explicitly declared; then, when a class was referred to which had not been previously declared, a warning could be issued and the error found and corrected.

As for epsilon continuations, it is theoretically possible to declare them within formative data files as empty-string formatives, in a manner analogous to the way they are defined in *lexc*; but as was pointed out in Section 3.1, epsilon continuations are not formatives, and it would be inelegant at best to treat them as formatives. They are more appropriately viewed as properties of formative classes than as members of those classes. Given that epsilon continuations are distinct entities from formatives, it would make sense to have a mechanism for defining epsilon continuations which is distinct from the mechanism for defining formatives; and as they are to be treated as properties of formative classes, it would make sense for epsilon continuations to be defined in conjunction with the definitions of the classes to which they belong.

So, the desiderata for declaring formative classes and epsilon continuations are:

- Require explicit declaration of formative classes
- Define epsilon continuations in a fundamentally different way than one defines formatives
- Ensure that formative class declarations and epsilon continuation definitions are handled within a single framework

Formative class declarations and epsilon continuation definitions are found in the file `data/classes.txt`. The syntax is simple. Each formative class declaration occupies a single line and begins with a colon, followed immediately by the name of the class, which must not contain spaces, and another colon. If the class has no epsilon continuations, the second colon is followed by a hyphen; otherwise, the epsilon continuations are listed, one per line, on the following lines. An epsilon continuation is simply the name of the formative class which is continued to, with no colons or other characters (whitespace characters before and/or after the class name are allowed). Blank lines are permitted, and non-blank lines may consist of or end with comments, which begin with an exclamation point and extend to the end of the line.

Example formative class declarations and epsilon continuation definitions are given in Figure 3.27 on the following page. The class `Root` includes (in fact, consists entirely of) an epsilon continuation to the class `Stem`, which has no epsilon continuations. The class `Noun Suffixes` has epsilon continuations to the classes `NominalPostbases`, containing all noun-attaching postbases, and `NounInflection`, containing all noun inflections; noun stems continuing to this class can then be suffixed with either a nominal postbase or a nominal inflectional ending. Neither `NominalPostbases` nor `NounInflection` has epsilon continuations. Class `EncliticsOrEnd` has epsilon continuations to the classes `Enclitics` and `Reduced` (containing reduced forms; see pages 34–36), as well as to the end-of-word class, `#`. `Enclitics` and `Reduced` have no epsilon continuations.

3.5 Declaration of Multicharacter Symbols

As described in Section 2.3.1 on pages 50–51, the fundamental unit of a finite-state machine is the symbol, and while by default single characters are treated as individual symbols, it is sometimes convenient for a symbol to be made up of several characters. In *lexc* (and therefore in the formats used in the Iñupiaq transducer's lexicon), entries are tokenized by the computer rather than by the morphologist, and thus the tokenizing algorithm needs a list of all the multicharacter symbols in the lexicon.

```

:Root:
  Stem

:Stem:-

:NounSuffixes:
  NominalPostbases
  NounInflection

:NominalPostbases:-

:NounInflection:-

:EncliticsOrEnd:
  Enclitics
  Reduced
  #

:Enclitics:-

:Reduced:-

```

FIGURE 3.27: Examples of formative class declarations and epsilon continuation definitions

The Iñupiaq transducer uses a large number of multicharacter symbols, which can be divided into two classes: automatically generated symbols and manually specified symbols. The automatically generated multicharacter symbols are used as rule triggers for allomorph filtering (see Section 3.3 on pages 97–98); the process which automatically generates these symbols (see Section 3.7.5 on pages 113–114) also produces a list of these symbols for the tokenizer. However, for the manually specified symbols (examples are given below), there is no process which automatically generates a listing; instead, this listing must be made by hand. The file `data/multichar_symbols.txt` exists for this purpose.

The format of this file is very simple: one multicharacter symbol per line; blank lines and comments (which begin with an exclamation point) are also permitted. Although comments are optional, if used judiciously they add significant value to the multicharacter symbol list, by allowing the list to become the primary source of documentation on the symbols used.

Figure 3.28 on the next page gives example entries from `multichar_symbols.txt`. Three types of multicharacter symbol are present in the list: flag diacritics, which begin and end with the `sign`; grammar tags (plus the `+Guess` tag, marking stems as guesses not necessarily listed in the lexicon), which begin with the `+` sign; and rule triggers, which consist entirely of capital letters. As the comments in the figure describe each multicharacter symbol, no additional commentary will be given here.

! for guessers
 GUESSNOUNSTEM ! dummy for noun guesser
 GUESSIVERBSTEM ! dummy for intransitive verb guesser
 GUESSTVERBSTEM ! dummy for transitive verb guesser
 +Guess ! distinguishes a guess from a known stem

! Morphophonological processes
 ABSDUAL ! the form of the absolute dual (and other suffixes
 ! identical in form to the absolutive dual) is conditioned on the phonology of
 ! the final syllable in the stem; the rules for this are defined in the XFST file
 OPTGEM ! optional gemination, conditioned by many inflectional morphemes
 GEM ! obligatory (when possible) gemination, conditioned by many inflectional
 ! morphemes
 HISTORICCONSONANT ! indicates that the preceding consonant was historically
 ! present, but has no phonetic realization unless geminated
 HISTORICSTOP ! indicates that the preceding fricative was a stop in
 ! proto-Eskimo (these go back to stops when they geminate, e.g.
 ! amaḡuq 'wolf' -> amaqquk 'two wolves')

! Combination patterns (see the abridged dictionary, p. xi)
 PLUS ! pattern represented in the dictionary by the plus sign
 MINUS ! pattern represented in the dictionary by the minus sign
 COLON ! pattern represented in the dictionary by the colon
 DIVISION ! pattern represented in the dictionary by the division sign
 PLUSOVERMINUS ! pattern represented in the dictionary by the plus-over-minus sign
 MINUSOVERPLUS ! pattern represented in the dictionary by the minus-over-plus sign

! Flag diacritics
 P.VALENCE.INTR ! set valence to intransitive
 P.VALENCE.TR ! set valence to transitive
 R.VALENCE.INTR ! require intransitive
 R.VALENCE.TR ! require transitive
 C.VALENCE ! clear valence
 U.NUMBER.SG ! set number to singular
 U.NUMBER.DU ! set number to dual
 U.NUMBER.PL ! set number to plural

! Grammar tags
 +V ! verb
 +Ind ! indicative mood
 +Prs ! "present"
 +Pst ! "past"
 +Int ! interrogative
 +Cont1 ! contemporative 1
 +Real ! realized
 +Unreal ! unrealized
 +Opt ! optative

FIGURE 3.28: Examples of multicharacter symbol declarations

3.6 Conversion of Lexical Data to XML

As an intermediate step during the conversion of the transducer's lexical data from the bespoke formats in which the data is entered to the XFST formats from which it can be compiled into a transducer, an XML-based representation of the lexicon is produced. While this can be justified on many grounds, not the least of which is Kenneth Beesley's endorsement of XML-encoded lexical data (see Beesley 2004*a, c*, 2003), the motivation for this arguably unnecessary step is historical. The original intended purpose of the Iñupiaq transducer was to provide lexical analysis for a machine translation system. As machine translation systems and lexical transducers both require a lexicon, it made sense for the lexicon to exist in an easily exchangeable format, a bill which XML fits very nicely. My original intention was to enter data as quickly as possible, convert this data to XML once and for all, and use the XML representation as the base lexical data for both morphological analysis and machine translation. However, the task of data entry seemed never to end, and I found I preferred to edit the original data files rather than to add to an XML representation of the data. Gradually, the focus of the project shifted away from machine translation, and it is unclear at present whether anyone will have any use for the XML representation of the lexicon, but since the machinery is already in place to generate it (and then to use it as the basis for generating the lexc representation of the lexicon), there is no compelling reason to remove it.

Although I hope that an XML-encoded version of the transducer's lexical data may be useful for other applications, no special accommodations have been made for any application beyond the Iñupiaq transducer. The XML representation of the data includes the contents of the files `stems.txt`, `postbases.txt`, `inflections.txt`, `enclitics.txt`, `patterns.txt`, `classes.txt`, and `multichar_symbols.txt` (all in the data directory). The file `data/inflections.txt` is converted by the Tcl script `generate_inflections.tcl`; the other files are converted by the Tcl script `lexicon_to_xml.tcl`.

3.6.1 Structure of the Resulting XML Document

Figure 3.29 on the following page gives a schematic of the structure of the XML-encoded lexicon. Element names are in bold; to the right of these is a number in parentheses, indicating how many elements of this type may be present in the document. If the element has attributes, their names (in italics) and data types (in small caps) are listed to the right of the element name and number. Question marks at the beginning and end of an attribute name indicate that the attribute is optional; the question marks are not part of the attribute name. Allowable children of a given element are listed below the parent element and indented one level; thus, a lexicon element may have the following child elements: `multicharSymbols`, `conditions`, `conditionXFST`, `class`, and `morpheme`.

The root element of the document is named `lexicon`; this element has one child element named `multicharSymbols`, one child element named `conditions`, one child element named `conditionXFST`, one or more child elements named `class`, and one or more child elements named `morpheme`. The `lexicon` element has no attributes.

The elements `multicharSymbols`, `conditions`, and `conditionXFST` are required nodes which serve as parents for optional nodes `multicharSymbol`, `condition`, and `xfstInstruction`, respectively. Each `multicharSymbol` element contains a single multicharacter symbol in its *symbol* attribute. Each `condition` element contains

lexicon (1)	
multicharSymbols (1)	
multicharSymbol (0+)	<i>symbol</i> STRING
conditions (1)	
condition (0+)	<i>name</i> STRING <i>pattern</i> STRING <i>oppositepattern</i> STRING
conditionXFST (1)	
xfstInstruction (0+)	<i>code</i> STRING
class (1+)	<i>name</i> UNIQUE ID
epsilon (0+)	<i>class</i> CLASS_NAME
morpheme (1+)	<i>abstractform</i> STRING <i>?gloss?</i> STRING
belongsto (1+)	<i>class</i> CLASS_NAME
continuation (1+)	<i>class</i> CLASS_NAME
variant (1+)	<i>condition</i> CONDITION_NAME <i>surfaceform</i> STRING <i>?strict?</i> 0 1

FIGURE 3.29: Schematic of the structure of the XML file containing the lexicon

a conditioning pattern name (in the *name* attribute), the xfst expression implementing that pattern (in the *pattern* attribute), and the xfst expression implementing the opposite pattern (in the *oppositepattern* attribute). Each *xfstInstruction* element contains one line of xfst code from SECTION XFST of *patterns.txt* (see pages 101–102).

Formative classes are listed in the *name* attribute of *class* elements. Formative classes with epsilon continuations have epsilon child elements; the name of the continuation class is given in the *class* attribute.

Formatives are listed in *morpheme* elements. These have one obligatory attribute, *abstractform*, which contains the underlying form of the formative. They also have an optional attribute, *gloss*, which may contain an English-language gloss for the formative. (It’s important to note that the presence of a *gloss* attribute may not necessarily indicate the presence of a gloss, as this attribute’s value may be an empty string.) Each *morpheme* must have at least one child element of each of the following types: *belongsto*, indicating in its *class* attribute a formative class to which the formative belongs; *continuation*, indicating (again, in its *class* attribute) a continuation class for the formative; and *variant*, indicating a “surface” representation of the formative (possibly an allomorph) as it was given in the formative data file where it was defined (in the *originalform* attribute) and as it will be passed to *lexc* (in the *surfaceform* attribute), the pattern name which conditions this particular surface form (in the *condition*, and, optionally, whether or not the form is a “standard” (i.e., non-innovative, non-archaic) form (in the *strict* attribute). At present, the *strict* attribute is used only for inflectional endings (see page 92), and is ignored at all further stages in the compilation of the lexicon. While the concept behind it deserves to be developed further, I consider it a mistake to have limited it to a (somewhat artificial) strict/non-strict dichotomy and to inflectional endings only. For that reason, some future version of the transducer will probably do away with the *strict* attribute.

Some readers may have noticed that there is an asymmetry between the *multicharSymbol*, *condition*, and *xfstInstruction* elements on the one hand, and the *class* and *morpheme* elements on the other. The former occur under an umbrella node whose child nodes are all of one type, whereas the latter are direct children of the root node. There is no real justification for this asymmetry, but as has no detrimental consequences to users of the file, it will be left as is.

3.6.2 Modifications Made to Data During Conversion to XML

The processes which convert the lexical data to XML also make a number of changes to the “raw” data from the source files to prepare it to be formatted as lexc/xfst code:

- Conditioning patterns defined with the help of Tcl procedures are converted to pure xfst, although they still contain the special symbol %s (see Section 3.3.2 on pages 101-104)
- Duplicate stems are identified and removed from the lexicon (at present, postbases and enclitics are not checked for duplicates)
- “Prefixes” specified in the inflectional ending file are attached to the suffixes with which they belong (see pages 88-92)
- For each formative belonging to a class that imposes specific long-distance dependency restrictions, an appropriate flag diacritic is appended or prepended to the formative string (see pages 70-71, 77-78, and 86-87)
- Formative representation is converted from “MacLean orthography” (see page 70) to a slightly different format used within the transducer

Most of these changes need no further explanation, but the transducer-internal orthographic conventions deserve to be elaborated on. As with Langgård and Trosterud’s transducer, the character ⟨e⟩ is used to represent “weak” *i* (/i/), while the character ⟨i⟩ is reserved for “strong” *i* (/i/), and the character ⟨l⟩ (U+013E) continues to be used in place of ⟨ł⟩. Additionally, the character ⟨q⟩ (U+024B) is used to represent “strong *q*” (/q/), and the character ⟨é⟩ is used to represent an “invariant” *i*, that is, one whose vowel quality does not change in a vowel cluster (this is necessary for postbases *-anik-* ‘previously, already’ and *-aqsi-* ‘about to; beginning to’, which preserve the default surface quality of a preceding *i*; see page 12). For all phonemes other than those just mentioned, the graphemes of the standard orthography are used.

Additionally, the characters which denote suffix attachment patterns (see Table 3.3 and discussion on pages 79-80) are replaced with multicharacter rule trigger symbols. Most of the suffix attachment symbols used in the lexical data files have a special meaning in xfst; replacing them with otherwise meaningless multicharacter symbols protects against the possibility of accidentally entering the characters in a rule as xfst operators rather than literal symbols. The mapping between the symbols used in the lexical data files and the xfst/lexc files is shown in Table 3.5 on the following page.

3.7 Conversion of XML Data to XFST Formats

The final step in the production of the Iñupiaq transducer’s lexicon is conversion from XML into lexc (for the lexical data) and xfst (for the filtering code used for handling allomorphy). This is carried out by `xml_to_lexc.tcl`, which is located in the scripts directory. In order to perform the conversion, this script must address the issue of formatives belonging to and/or continuing to more than one class, since a given entry in a lexc LEXICON can only belong to that class and can only continue to one class. Also, the script must generate xfst expressions for “otherwise” conditions (see pages 100-101).

TABLE 3.5: Suffix attachment symbols in the lexical data files and in xfst/lexc files

In lexicon data files	In xfst/lexc files
-	MINUS
+	PLUS
+-	PLUSOVERMINUS
-+	MINUSOVERPLUS
/	DIVISION
:	COLON
=	EQUAL
~	TILDE

3.7.1 Resolution of Membership in More Than One Class

Some inflectional suffixes in the Iñupiaq transducer are declared to belong to more than one formative class, but in lexc a formative can only belong to one LEXICON at a time. Of course, there is no prohibition against putting identical copies of a formative into multiple LEXICONS, but this is discouraged (see Beesley and Karttunen 2003:243–245); it would be preferable to list any given formative exactly once, if only for the sake of elegance. In the case of formatives with membership in more than one class, this can be accomplished with epsilon continuations. A number of solutions are possible; the one I adopt is illustrated in Figure 3.30. Formatives with membership in more than one class (in the figure, *suffix 3* and *suffix 4*, which belong to both Class A and Class B) are removed from the classes to which they belong, and placed in a new class (in the figure, this is Class C). Epsilon continuations are then added to each of the classes to which they originally belonged; in this way, the members of the new class are in effect members of their former classes as well, but they are only listed once in the file.

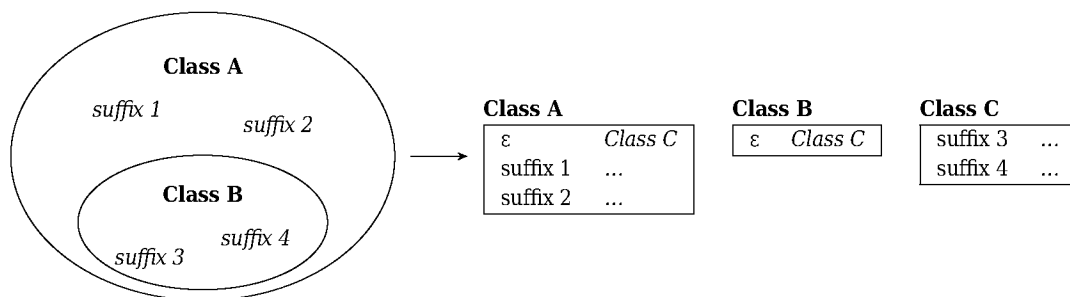


FIGURE 3.30: Treatment of formatives with multiple class memberships in the Iñupiaq transducer data model (left of the arrow) and in the lexc data model (right of the arrow)

As Class B in Figure 3.30 is entirely subsumed by Class A, a more optimal conversion to lexc would

be to leave *suffix 3* and *suffix 4* in Class B and include an epsilon continuation from Class A to Class B; this solution is illustrated in Figure 3.31. As it turns out, for all formatives in the Iñupiaq transducer’s lexicon which belong to two classes, one of the two classes is a proper subset of the other, and thus a Figure 3.31-style solution would have more elegantly described the lexicon. However, the two solutions are functionally equivalent, and the two lexc representations will compile to identical transducers, so I see no compelling need to switch from the solution currently in place to the more concise one.

Class A		Class B											
<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 2px 10px;">ϵ</td> <td style="padding: 2px 10px;"><i>Class B</i></td> </tr> <tr> <td style="padding: 2px 10px;">suffix 1</td> <td style="padding: 2px 10px;">...</td> </tr> <tr> <td style="padding: 2px 10px;">suffix 2</td> <td style="padding: 2px 10px;">...</td> </tr> </table>	ϵ	<i>Class B</i>	suffix 1	...	suffix 2	...		<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 2px 10px;">suffix 3</td> <td style="padding: 2px 10px;">...</td> </tr> <tr> <td style="padding: 2px 10px;">suffix 4</td> <td style="padding: 2px 10px;">...</td> </tr> </table>	suffix 3	...	suffix 4	...	
ϵ	<i>Class B</i>												
suffix 1	...												
suffix 2	...												
suffix 3	...												
suffix 4	...												

FIGURE 3.31: A more optimal lexc treatment of the formatives in Figure 3.30

3.7.2 Resolution of Continuation to More Than One Class

The case of formatives which continue to more than one class is similar to that of formatives which belong to more than one class: lexc provides no direct support for these cases, but they can be modeled with the help of epsilon continuations and additional formative classes. For formatives with multiple continuations, the conversion into lexc is accomplished via the following algorithm:

1. Let F equal the set of formatives which continue to more than one formative class.
2. Let C equal the set of unique sets of formative classes such that for any set c in C there exists at least one formative f in F which continues to each class in c and to no other classes.
3. For each c in C :
 - a) create a new formative class n consisting entirely of epsilon continuations to each class in c ;
 - b) let F_c equal the subset of F such that each f in F_c continues to each class in c and to no other classes;
 - c) for each f in F_c :
 - i. remove all continuations;
 - ii. create a single continuation to n .

A simple, hypothetical input to this algorithm and the corresponding output are shown in Figure 3.32 on the next page. In this example, $F = \{\text{suffix 1, suffix 2}\}$; $C = \{\{\text{Class B, Class C}\}\}$; for the case of $c = \{\text{Class B, Class C}\}$, $n = \{\text{Class D}\}$ and $F_c = \{\text{suffix 1, suffix 2}\}$. In other words, *suffix 1* and *suffix 2* both continue to the same set of classes: Class B and Class C. A new class, Class D, is created and populated with epsilon continuations to Classes B and C; *suffix 1* and *suffix 2* are then made to continue to Class D.

3.7.3 Resolution of “Otherwise” Conditions

In the lexical files of the Iñupiaq transducer, all productive allomorphs of a morpheme are specified together, along with codes denoting the phonological environment which conditions each allomorph. Among

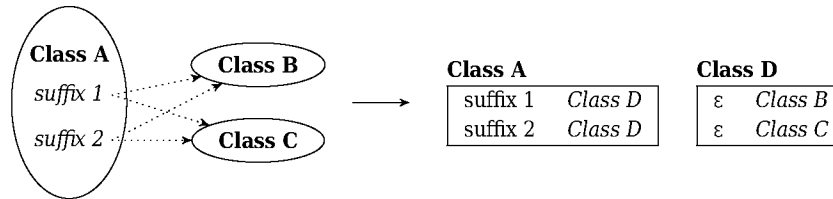


FIGURE 3.32: Treatment of formatives with multiple continuations in the Iñupiaq transducer data model (left of the solid arrow; dotted arrows represent continuations) and in the lexc data model (right of the solid arrow)

other things, this makes it possible to specify “otherwise” as one of the phonological environment codes, indicating that this allomorph is to be used when the preceding environment does not match any of the conditions specified for the other allomorphs. xfst, however, has no concept of “otherwise,” and therefore the exact parameters of all such conditions must be expressed in a self-contained manner.

As discussed in Section 3.3, for each allomorph conditioning environment specified in data/patterns.txt, two expressions are defined: the main expression, which accepts strings containing the environment in question, and a ‘complement’ expression which accepts strings not containing the environment (see pages 102–104). The purpose of the second definition is to facilitate computation of “otherwise” conditions. For a set of allomorphs with non-“otherwise” conditions, the corresponding “otherwise” condition can be expressed as the intersection of the complement expressions of all conditions in the set. This is a trivial operation in xfst.

3.7.4 Outputting the Lexicon in lexc Format

Having carried out the steps outlined in Sections 3.7.1 and 3.7.2, the modified lexicon now complies with the limitations of lexc, specifically in that each formative belongs to only one class and continues to only one class. As such it can be expressed in lexc format.

The process of generating lexc output is quite simple. The output begins with a comment, which lexc will ignore, indicating the name of the script that generated the file and the date and time when it was generated. Next, a list of multicharacter symbols is given (see pages 50–51). Finally, for each formative class in the lexicon, the name of the class is given and each member of the class (including epsilon continuations) is listed along with its continuation class.

3.7.5 Outputting Allomorph Conditioning Environment Data in xfst Format

After exporting the formatives in lexc format, the final step in the conversion of the Iñupiaq lexicon into XFST formats is to output allomorph conditioning environment data. This file, ipk_condition_xfst.txt, contains the contents of SECTION XFST from the file data/patterns.txt (see pages 101–102), the definition of an xfst variable named ConditionSymbols (which lists the names of each rule trigger symbol), and the definitions of each conditioning environment filter. Following these, a variable named LexiconCheckConditions is defined; it is a cascade of variables composed together, beginning with a variable named Lexicon, followed by each variable containing a conditioning environment filter definition, and terminated by the rule

[ConditionSymbols -> 0], which deletes all rule triggers associated with conditioning environments. Finally, the variable `Lexicon` is redefined to the value of `LexiconCheckConditions`. The use of the variable `Lexicon` and variables like `LexiconCheckConditions` is explained in Section 3.8.1 on pages 114–116.

3.8 Morphophonological Rules

The main purpose of the morphophonological (or more correctly, morphographemic) rules in this project is to transform the data contained in the lexicon into a mapping between, on one hand, strings of morphemes in some citation or “underlying” form, and on the other hand, surface forms of Iñupiaq words. The lexicon, as it exists just before rules are applied to it, is a transducer whose upper and lower languages are in intermediate states relative to the eventual upper and lower languages of the finished transducer. These intermediate upper and lower languages are both abstract. The upper language contains grammar tags for inflectional morphemes, while the lower language contains actual alphabetic forms of these morphemes. Both upper and lower languages contain extralinguistic symbols such as morpheme boundary markers, rule triggers, and flag diacritics. Both use a non-standard orthography (see page 110). On both sides, rule triggers will need to be removed and spellings brought into conformance with standard Iñupiaq conventions. Additionally, on the lower side, entries containing allomorphs in disallowed environments will need to be filtered out of the transducer; a number of morphophonological changes will need to be made to produce the appropriate surface forms; and non-alphabetic symbols such as morpheme boundaries will need to be removed. All of these changes will be brought about by morphographemic rules composed together with the lexicon.

A secondary purpose of the rules in the Iñupiaq transducer is to define the set of phonotactically possible noun and verb stems. In the event that a word cannot be fully analyzed according to the data in the lexicon, possible stems can be generated from these rules, and the remainder of the word analyzed according to the postbases, inflectional endings, and enclitics in the lexicon.

Unlike the lexicon, the morphophonological rules are not encoded in a format specific to this project; they are written directly in `xfst`. Readers unfamiliar with `xfst` may wish to consult Appendix A on pages 168–170 to better understand the regular expressions and other `xfst` code in this section.

3.8.1 Architecture and Idioms

There are several possible ways to structure an `xfst` file, depending on one’s goals and personal preferences. The architecture of the morphophonological rule file for the Iñupiaq transducer has evolved dramatically over the course of its development. Initially, I adopted a format very similar to the one used by Langgård and Trosterud: a section of symbol class definitions, followed by a section of rules, followed by a cascade in which the rules are composed into a single transducer. But because my transducer shifts more of the descriptive burden onto the rule component than does theirs, I found this structure suboptimal for my needs in two respects.

First, as the rules became more numerous (and compile time increased accordingly), it became increasingly important to have feedback from XFST not only after the transducer was compiled, but throughout the compilation process. When XFST evaluates a rule file, it displays information on the results of each command as soon as that command is executed. Composing all the rules together via a single cascade

(in other words, a single xfst command) prevents any feedback until the entire composition process is complete.

Second, when debugging one's xfst rules, it is often useful to examine the output of a transducer composed of the lexicon and all the following rules up to a certain point. If the complete transducer was built using a single rule cascade, then creating a transducer with only some of the rules requires that another rule cascade, containing only the rules up to the point of interest, be written and executed. This requires XFST to repeat work that it has previously performed: in compiling a transducer built from a lexicon composed with rules $0\dots n$, XFST must necessarily build every transducer consisting of the lexicon composed with rules $0\dots n - m$, $0 \leq m \leq n$ (as well as compiling the transducer consisting of the lexicon alone).

To address these shortcomings, I have adopted an architecture where the final transducer is built up in pieces, with a trail of "crumbs" containing the intermediate transducers compiled along the way. Related commands are grouped together (for example, all commands defining assimilation). Groups which define rules end in a pair of variable definitions. The first defines a "crumb" variable containing the existing transducer composed with the new rules; the contents of these variables can easily be loaded into XFST for debugging, saving users from having to re-compose the part of the transducer they wish to examine. The second command in the pair redefines the variable `Lexicon` to contain the same transducer as the crumb variable just defined. In this way, `Lexicon` becomes a relative point of reference, while the crumb variables are absolute points of reference. The definitions of the crumb variables take advantage of this fact, composing new material to the value of `Lexicon` rather than to the value of the previous crumb variable. Because rules are added relative to the current state of the transducer, re-ordering them is simply a matter of cutting and pasting the group of commands into the appropriate place; no changes need be made to the code which performs the composition.¹⁵

Three other idioms are used regularly throughout the rules. The first is the definition of "convenience" variables, which do not contain rules but rather patterns (such as the structure of a permissible consonant cluster) or classes of symbols (such as the set of voiced consonants) which make rules easier to write and read. A set of convenience definitions with widespread application is provided at the beginning of the file, and other convenience definitions with more limited scope are defined with the groups of commands where they are used.

The second idiom, which unfortunately tends to obfuscate the rules somewhat, is "ignoring." Previously discussed in Section 3.3.1 on pages 98-101, an ignoring expression takes the form `[A / B]` and results in a pattern such that zero or more instances of pattern B may occur at any point within or on either edge of pattern A. This is useful for dealing with rule triggers, morpheme boundary markers, and other special symbols which are peppered throughout the lexicon. Rules generally operate on alphabetic symbols, and while a given rule may be sensitive to one special symbol, all other non-alphabetic symbols which may be present will be irrelevant. Yet the rule must account for the presence of those irrelevant symbols or it will fail.¹⁶ As the number of rule triggers and other special symbols increases, so does the difficulty

¹⁵This could alternatively be accomplished by pushing each crumb onto the xfst stack (see Appendix A) and popping it back off in place of referring to `Lexicon` in each rule definition. The choice between these two approaches is a matter of taste.

¹⁶Flag diacritics are particularly insidious; they occur all over the lexicon, rules are never sensitive to them, and they are normally invisible to users interacting with XFST. Bugs due to rules not taking the presence of flag diacritics into account are maddeningly hard to find. Because they are so much trouble, I prefer to filter them completely out of the transducer rather than trying to anticipate

of anticipating which triggers might occur where, in which order, within a rule. Rather than target each symbol individually, one might define a variable containing an inclusive disjunction of all such symbols, and invoke this disjunction, followed by a Kleene star, at points in a rule where these symbols might occur (in other words, one could indicate that at specified places within a rule, zero or more rule triggers may be present). But even then, it is easy to forget to mark a point where non-alphabetic symbols might occur, and such an omission would be hard to identify. Ignoring represents the ultimate lazy/safe approach to irrelevant symbols. Rather than specify points where irrelevant symbols may occur (and risk missing points here and there), ignoring specifies that irrelevant symbols might occur anywhere in the expression.

The specification of symbols to ignore also deserves a brief explanation. Rather than explicitly list the set of symbols which are irrelevant for a given rule, I calculate the term complement (represented in xfst by a backslash `\`) of the set of symbols which *are* relevant for that rule. Alphabetic symbols are almost always relevant—they bear on whether most rules should fire or not. For many rules, a particular rule trigger or a morpheme boundary may also be relevant. The term complement of the set of relevant symbols is the set of all single symbols not in the set of relevant symbols—in other words, the set of irrelevant symbols. Defining the set by complementation rather than by explicit enumeration also reduces the likelihood of having to rewrite a number of rules in the event that a new non-alphabetic symbol is introduced into the lexicon.

The third idiom might be called “filtration through complementation.” It is used less frequently than the other two idioms but frequently enough to deserve a brief mention. It involves expressions of the form `~$A` where *A* is a subexpression describing a regular language. The dollar sign is the containment operator; it takes a subexpression as its operand and describes a language¹⁷ where every member string contains at least one substring which is defined by the subexpression. The tilde is the language complementation operator; it takes as its operand a subexpression defining a regular language (not a regular relation) and defines the regular language which is the complement of the language described by its operand. In other words, it defines a language that contains no string in the operand’s language and every string not in the operand’s language. When composed “below” a transducer, an expression of the form `~$A` filters out any strings in the lower language which contain a substring defined by *A*, as well as any strings in the upper language which exist in a one-to-one relationship with lower-language strings which the expression has filtered out. When composed “above” a transducer, an expression of that form filters out strings in the upper language, as well as any strings in the lower language which exist in a one-to-one relationship with them.

3.8.2 The Rules

3.8.2.1 Loading the Lexicon; Eliminating Flag Diacritics

The rule code begins as shown in Figure 3.33 on the next page. The lexicon, as prepared by the script `xml_to_lexc.tcl` (see Section 3.7 on pages 110–114), is loaded onto the top of the XFST stack. The lexicon is then restructured using the `eliminate flag` command so that the restrictions that had been imposed by the

their presence as I do with all other special symbols. See Section 3.8.2.1 on pages 116–117.

¹⁷The containment operator can also describe regular relations, but as the `lfupiaq` transducer never uses it for that purpose I have omitted it from the discussion.

flag diacritic mechanism are instead imposed by the structure of the transducer itself. This is a relatively expensive operation, causing the transducer to more than double in size, which must inevitably slow down all subsequent steps in the compilation process. The payoff is the peace of mind that comes from knowing that subsequent rules need not take the presence of these symbols—which are normally invisible during interactive XFST sessions—into account.

```
read lexc ipk_lexc.txt
eliminate flag VALENCE
eliminate flag NUMBER
define LexiconNoFlags;
define Lexicon LexiconNoFlags;
```

FIGURE 3.33: Code to load the lexicon and eliminate flag diacritics

When the lexicon has been restructured, it is popped off the XFST stack using the command `define LexiconNoFlags`, the first crumb variable definition in the file. Finally, the variable `Lexicon` is assigned its initial value.

3.8.2.2 Global Convenience Definitions

In this section, shown in Figure 3.34 on the following page, a number of symbol classes are defined for use in subsequent rules. Because `xfst`'s variable mechanism allows an expression to be associated with a meaningful name, convenience definitions help make other expressions more readable, and because the same definition can be invoked multiple times, variables also make it easier to write expressions.

Many of the classes defined constitute graphemic equivalents of phonological natural classes. Others are classes of non-alphabetic symbols. The class `Graphemes` will be used to create expressions of symbols for rules to ignore (see preceding page). All but one class is defined through logical disjunction (also called union). `ConsonantCluster` is defined through subtraction (also called set-theoretic difference or relative complement), together with ignoring; it defines the set of strings consisting of two substrings from the `Consonant` class, but excluding the strings `{kh}`, `{qh}`, and `{sr}`, which are digraphs and therefore single consonants, despite the fact that `{h}`, `{k}`, `{q}`, `{r}`, and `{s}` are members of the `Consonant` class by themselves.

```

define Vowel [a | e | é | i | u]; ! i = strong i; e = weak i; é = weak i which does not become a when followed by another vowel
define AnyQ [q | q]; ! using q to represent strong Q
define Stop [p | t | c | h | k | AnyQ];
define VoicelessFricative [ʃ | ʀ | s | s r | k h | q h | h];
define VoicedFricative [v | l | l | y | r | g | ĝ];
define Nasal [m | n | ñ | ŋ];
define VoicedConsonant [VoicedFricative | Nasal];
define VoicedPhoneme [VoicedConsonant | Vowel];
define Consonant [VoicedConsonant | VoicelessFricative | Stop];
define Graphemes [ a | b | c | d | e | é | f | g | ĝ | h | i | j | k | l | l | l |
  | ʀ | m | n | ñ | ŋ | o | p | q | q | r | s | t | u | v | w | x | y | z ];
define ConsonantCluster [ [ Consonant Consonant ] - [ k h | q h | s r ] ] / \Graphemes;

define WordBoundary [ %< | .# . ];
define MorphemeBoundary [ %> | WordBoundary ];
define MorphemeBoundary2 [ MorphemeBoundary | %| ];

define CombinationPattern [ WEAKENI | PLUS | MINUS | COLON | DIVISION |
  PLUSOVERMINUS | MINUSOVERPLUS | EQUAL | TILDE | UVULARIZE | VELARIZE |
  PROTECTWEAKI | LENGTHENV ];
define DeleteOnTop [ NONSTEM | OPTNSTEM | HISTORICCONSONANT | HISTORICSTOP | YTCH
  | REGRESSIVEPALATALIZATION | CombinationPattern ];

```

FIGURE 3.34: Global convenience definitions

3.8.2.3 Limiting Word Length

The first rule in the file, given in Figure 3.35 on the following page, imposes some sane upper limits on the length of words that may be recognized by the transducer. The primary reason for including this rule was to simplify development. XFST can print random words from the upper or lower language of a transducer, but this feature is only helpful if words are reasonably short. However, the restrictions imposed by this rule seem reasonable and few if any real-world Inupiaq words will exceed them. These restrictions are:

- words must contain fewer than eight within-word morpheme boundaries, effectively limiting words to a maximum of eight within-word morphemes (an initial morpheme plus a morpheme following each boundary)
- words must contain fewer than three word-enclitic boundaries, effectively limiting words to a maximum of two enclitics (one after each boundary)
- words must contain fewer than three word-reduced-form boundaries, effectively limiting words to a maximum of two suffixed reduced forms
- words must contain fewer than eight total boundaries of any kind, effectively limiting words to a maximum of eight total morphemes

Ignoring and complementation are used to disregard any non-boundary characters.

Similarly to the elimination of flag diacritics, this rule imposes considerable size and performance penalties, particularly in its cumulative effects on the rest of the compilation process. For some possible applications of the transducer, the benefits of this rule may not justify the drawbacks. The limitations of

```

define LimitWordLength [
  [%> / \%> ] ^ < 8 .o.
  [%< / \%< ] ^ < 3 .o.
  [%| / \%| ] ^ < 3 .o.
  [ [ %> | %< | %| ] / [ \%> | \%< | \%| ] ] ^ < 8 ;
define LexiconLimitedWordLength [ Lexicon .o. LimitWordLength ]
define Lexicon LexiconLimitedWordLength;

```

FIGURE 3.35: Code to limit word length

human cognition tend to naturally restrict word length, so if the transducer is to be used in an application where it will only process authentic, human-generated Iñupiaq words, it may make sense to compile the transducer without this rule.

3.8.2.4 Eliminating Capital Letters from the Lexicon

The code in this section ensures that the lexicon contains no capital letters, in order to make the transducer simpler to use. In Figure 3.36, the code has been truncated. It should contain two subexpressions for each letter of the alphabet: a left replacement subexpression composed above the lexicon, and a right replacement subexpression composed below it. The left replacement subexpressions remove capital letters from the upper language, and the right replacement subexpressions have the same effect on the lower language.

```

define LexiconNoCaps [
  [ a <- A ] .o.
  [ b <- B ] .o.
  ...
  [ z <- Z ] .o.
Lexicon .o.
  [ A -> a ] .o.
  [ B -> b ] .o.
  ...
  [ Z -> z ] ];
define Lexicon LexiconNoCaps;

```

FIGURE 3.36: Code to limit word length

3.8.2.5 Defining Stem Guessers

The code in this section defines two rules for guessing stems (see Section 2.3.7 on pages 59–60), one for nouns and one for verbs, and inserts this code into the transducer. The rules are based on Iñupiaq phonotactic constraints (see Section 2.1.3 on pages 25–27); because of this, they will probably fail on words of foreign origin, but are likely to produce higher-quality guesses for words of Iñupiaq origin which,

for whatever reason, are not in the lexicon of the non-guessing transducer.¹⁸

The first step involves defining a series of convenience variables, each one addressing a different aspect of Iñupiaq stem phonotactics. These definitions are given in Figure 3.37 below. The most complex aspect of the rules is defining possible stem-medial consonant groups; there is also a simple definition of allowable stem-initial consonants and an even simpler definition of legal vowel groups (one /i/ vowel or one or two “prime” vowels). All of these definitions are consolidated in the variable `StemWithoutFinalConsonant`, which is defined as an optional initial consonant, followed by a vowel group, followed by zero or more consonant group/vowel group pairs. Finally, three filters are defined: `EnforcePalatalRestrictions` eliminates guesses where /i/ is followed by a non-palatalized consonant or where /i/ is followed by a palatalized consonant; `EnforceWordInitialDistribution` throws out guesses beginning with /ti/, /ni/, or /si/; and `EnforceQDistribution` removes guesses ending in /iġ/ or /VCV₁(V₂)ġ/ where V₁ and V₂ are vowels other than /i/. Each of the filters use the “filtration through complementation” idiom described on page 116.

```
define InitialConsonant [ ( [ p | m | t | n | s | y | k | q ] ) ]; ! parentheses are included because initial consonants are optional
define StopCluster [ p p | p t | p k | p q | t t | t p | t k | t q | t c h | k k | k p | k t | q q | q p | q t ];
define VoicelessFricativeCluster [ v f | v f | v s | v s r | v k | v q | f f | f h | f h | f r | f r | s k |
  s s | s t | k s | p s | q s | r s | k s r | q s r | t s r | k f | k f | q f | q f ];
define VoicedFricativeCluster [ v v | v l | v l | v y | v r | v g | v ġ | l v | l l | l y | l r | l g |
  l ġ | l v | l l | l y | l r | l g | l ġ | y v | y l | y l | y r | y g | y ġ | r v | r l | r l | r y | r r |
  r g | r ġ | g v | g l | g l | g y | g r | g g | ġ v | ġ l | ġ l | ġ y | ġ r | ġ ġ ];
define NasalCluster [ g m | g n | g ñ | ġ m | ġ n | ġ ñ | m m | m n | m ñ | m ŋ | m l | m l | m g | m ġ |
  n m | n n | n ŋ | n l | n g | n ġ | ñ m | ñ ñ | ñ ŋ | ñ l | ñ g | ñ ġ | ŋ m | ŋ n | ŋ ŋ | ŋ l | ŋ l | ŋ g ];
define LegalConsonantCluster [ StopCluster | VoicelessFricativeCluster | VoicedFricativeCluster | NasalCluster ];
define StemMedialConsonantComponent [ Consonant | LegalConsonantClusters ];
define VowelComponent [ [ a | i | u ] ^ { 1, 2 } | e ];
define StemWithoutFinalConsonant [ InitialConsonant VowelComponent
  [ StemMedialConsonantComponent VowelComponent ] * ];
define EnforcePalatalRestrictions ~ $ [ i (Consonant) [ l | f | n ] | i t Vowel | [ Vowel - i ] (Consonant) [ ñ | l | r ] ];
define EnforceWordInitialDistribution ~ $ [ . # . [ t | n ] i | s e ];
define EnforceQDistribution ~ $ [ e q | Vowel Consonant [ Vowel - e ] ^ { 1, 2 } q ];
```

FIGURE 3.37: Convenience definitions for guessing stems

Next, the actual guessing rules are defined, as shown in Figure 3.38 on the following page. The two rules are nearly identical; the only differences are the specific consonants allowed to occur stem-finally, and the fact that noun stems are filtered for illegal occurrences of “strong” and “weak” {q}, a distinction which applies only to nouns.

With the rules defined, the final step is to insert them into the lexicon. The command `push Lexicon` puts the contents of the transducer onto the XFST stack so that this can happen. Three placeholder symbols are defined in the lexicon: `GUESSNOUNSTEM` is listed as a noun stem, `GUESSIVERBSTEM` is listed as an intransitive verb stem, and `GUESSTVERBSTEM` is listed as a transitive verb stem. The commands beginning

¹⁸Should there arise some compelling reason to recognize words for which these guessers fail, one could create a transducer built on an unscrupulous guessing rule which simply accepts any string ending in what might be an Iñupiaq inflectional suffix (including a zero morph). Words which fail the standard transducer and the phonotactics-based guesser could then be passed to this “transducer of last resort” for a final attempt at analysis.


```

define PossibleNounStem [
  [ StemWithoutFinalConsonant ([ k | q | q | n | ñ ]) "+Guess":0 ] .o.
  EnforcePalatalRestrictions .o.
  EnforceWordInitialDistribution .o.
  EnforceQDistribution ];
define PossibleVerbStem [
  [ StemWithoutFinalConsonant ([ k | q | t ]) "+Guess":0 ] .o.
  EnforcePalatalRestrictions .o.
  EnforceWordInitialDistribution ];

```

FIGURE 3.38: Rules for guessing noun and verb stems based on Iñupiaq phonotactics

with substitute defined ... replace these symbols with the guessing patterns defined in PossibleNounStem and PossibleVerbStem. The continuations associated with the placeholder symbols are now associated with the guessing patterns. The modified transducer is popped back off the stack and stored in a new crumb variable (define LexiconWithGuessing), whose contents are then copied into Lexicon.

```

! replacing the special symbols with guesser content
push Lexicon
substitute defined PossibleNounStem for GUESSNOUNSTEM
substitute defined PossibleVerbStem for GUESSIVERBSTEM
substitute defined PossibleVerbStem for GUESSTVERBSTEM
define LexiconWithGuessing
define Lexicon LexiconWithGuessing;

```

FIGURE 3.39: Code to insert the stem-guessing rules into the transducer

3.8.2.6 Filters for Irregular Inflected Forms

Occasionally, a noun stem will have a dual or plural form which cannot be correctly generated by the transducer's rules. For example, MacLean (n.d.a) gives the plural of *tuvaq* 'hunter' as *tuviggat*, whereas the transducer would generate **tuvaat*. To remedy this situation, the stem is specified in the file `data/stems.txt` as `tuvaqIRREGULARPLURAL n`, and two additional entries (one each for absolutive and relative case) describe the plural form: `tuviggat>+N+Abs+Pl:tuviggat *` and `tuviggat>+N+Rel+Pl:tuviggat *`. The rule shown in Figure 3.40 on the next page then filters the incorrect plural out of the transducer. The same process can be applied to stems whose dual forms the transducer cannot generate; the multicharacter symbol `IRREGULARDUAL` is appended to the stem in the file `data/stems.txt`, and separate entries are made for the actual dual form (one entry each for absolutive and relative cases).

The filters in this section use the "filtration through complementation" idiom described on page 116. The "ignoring" operator is not used because there is only one point in each filter where irrelevant symbols might intervene: directly after the first symbol. Instead of ignoring, a combination of term complementation (a backslash) and Kleene closure (an asterisk) are used.

```

define IrregularMorphologyFilter [ IRREGULARDUAL | IRREGULARPLURAL ]; ! convenience definition
define IrregularDualFilter [ ~$[ IRREGULARDUAL \[ IRREGULARDUAL | %> ]* %> "+N" ["+Abs" | "+Rel" ] "+Du" ] ];
define IrregularPluralFilter [ ~$[ IRREGULARPLURAL \[ IRREGULARPLURAL | %> ]* %> "+N" ["+Abs" | "+Rel" ] "+Pl " ] ];
define LexiconIrregularMorphology [
  0 <- IrregularMorphologyFilter .o.
  IrregularDualFilter .o.
  IrregularPluralFilter .o.
  Lexicon .o.
  IrregularMorphologyFilter -> 0 ];
define Lexicon LexiconIrregularMorphology;

```

FIGURE 3.40: Code to filter out unattested “regular” dual or plural members of a paradigm

The composition cascade is more complicated for these filters than for most of the other rules in the transducer. The cascade is bracketed on either end with expressions deleting the multicharacter symbols IRREGULARDUAL and IRREGULARPLURAL from the upper and lower languages. Between these expressions, the two filters are composed “above” the transducer because the filters are sensitive to grammar tags, which are only present in the transducer’s upper language.

3.8.2.7 Reducing Two Consecutive Boundary Markers to One

When a morphological word ending in a zero morph is followed by an enclitic or reduced form, the representation of that word in the lower language of the lexicon will contain two consecutive boundary markers with no symbols in between. For the sake of simplicity, the rule shown in Figure 3.41 gets rid of the first of these boundary markers.

```

define ReduceDoubleBoundary [ %> -> 0 || _[%< | %| ] ];
define LexiconNoDoubleBoundary [ Lexicon .o. ReduceDoubleBoundary ];
define Lexicon LexiconNoDoubleBoundary;

```

FIGURE 3.41: Code to fix strings of two consecutive boundary markers

3.8.2.8 Changing Noun-Stem-Final *-n* to *-ti*

At this point in the file, definition of morphographemic rules begins in earnest. The first rule deals with the alternation of [n] and /ti/ at the end of noun stems (see Section 2.1.2.7 on pages 22–23). Stems susceptible to this alternation are listed in the stem file with a final ⟨n⟩, despite the fact that [n] is only a surface form. Given that every occurrence of noun-stem-final surface [n] corresponds to an underlying /ti/, while not every underlying /ti/ at the end of a noun stem can correspond to surface [n], I found it easier to mark applicable stems by including ⟨n⟩ rather than ⟨ti⟩ in their definitions, and then to apply a rule converting this ⟨n⟩ to ⟨ti⟩ where appropriate, than to list stems as ending in ⟨ti⟩ plus a rule trigger and apply a rule converting these instances of ⟨ti⟩ to ⟨n⟩. As an added bonus, using ⟨n⟩ directly in stem listings

also made data entry easier, since all of the lexical sources for this project use the absolutive singular form as the citation form for nouns.

The definition of the rule is shown in Figure 3.42. Two rule triggers are relevant for this rule: `OPTNSTEM` indicates that, if the preceding stem exhibits the /tĩ/-[n] alternation, the surface realization of the stem in this particular word may end in either ⟨n⟩ or ⟨tĩ⟩; `NONSTEM` indicates that, if the preceding stem exhibits the /tĩ/-[n] alternation, the surface realization of the stem in this particular word must end in ⟨n⟩. The expression defined as `OptionalNStem`, when composed with `Lexicon`, will take the lower language of `Lexicon` and replace each string containing the symbol `OPTNSTEM` with two strings, one where this symbol is replaced with the symbol `NONSTEM`, and another where this symbol is simply deleted.

```
define OptionalNStem [ OPTNSTEM -> [NONSTEM | 0] ];
define NStem1 [ n -> t e, ñ -> s e || _ [%> \NONSTEM] / [\Graphemes | %> | NONSTEM] ];
! n becomes tĩ when followed by a (regular) morpheme boundary and the
! following morpheme doesn't prevent this change
define NStem2 [ n (->) t e, ñ (->) s e || _ [[%< | %] \NONSTEM] / [\Graphemes | %< | % | NONSTEM] ];
! n optionally becomes tĩ when followed by an enclitic or reduced form
! boundary and the following morpheme doesn't prevent this change
define NStem3 [ NONSTEM -> 0 ];
define LexiconNStem [ Lexicon .o. OptionalNStem .o. NStem1 .o. NStem2 .o. NStem3 ];
define Lexicon LexiconNStem;
```

FIGURE 3.42: Code defining the /tĩ/-[n] alternation

`NStem1`, composed under `Lexicon` and `OptionalNStem`, will replace all instances of ⟨n⟩ in the transducer's previous lower language with ⟨te⟩ (recall that ⟨e⟩ represents /i/ within the transducer) and all instances of ⟨ñ⟩ with ⟨se⟩ when either string is followed by a morpheme boundary and a symbol other than `NONSTEM`, ignoring symbols other than alphabetic symbols, interword boundary markers, and `NONSTEM`. `NStem2` will optionally replace ⟨n⟩ and ⟨ñ⟩ in the transducer's previous lower language with ⟨te⟩ and ⟨se⟩, respectively, when either of these strings is followed by an enclitic or reduced-word boundary and a symbol other than `NONSTEM` (again, ignoring irrelevant symbols). Finally, `NStem3`, composed to the bottom of the transducer, creates a new lower language which is identical to the previous one except that it does not contain the symbol `NONSTEM`.

3.8.2.9 Forming the Absolutive Dual

As was discussed previously, the practice I have adopted regarding allomorphy is to divide the labor between the lexicon and the rules. Allomorphic variants and their conditioning environments are specified in the lexicon (see page 80 and pages 87–92), and controls to ensure that stems and allomorphs are appropriately paired are defined with rules (specifically, conditioning environment filters; see Section 3.3 on pages 96–104). However, the absolutive dual is not handled in this way; in the lexicon, it is represented only by the multicharacter symbol `ABSDUAL`, and the realization of this symbol is accomplished using rules. The main reason for implementing the absolutive dual within the rules is that it forms the basis for a number of other dual suffixes, including all of the other unpossessed dual case endings and several possessive

ones. Implementing the absolutive dual as a rule and rule trigger makes possible morpheme specifications such as ABSDUAL/nik (the modalis dual ending) and ABSDUAL+kpiñ (one form of the relative dual with 2nd person singular possessor ending). If the absolutive dual were implemented in the same way as other inflectional morphemes, then each morpheme that is based on it would either have to include in its specification a duplication of all the absolutive dual allomorphs, or else be implemented using a continuation from the absolutive dual—a strategy that would be incompatible with the framework for specifying inflectional endings described in Section 3.2.3.2.

The form of the absolutive dual ending is *-k* in all cases, but depending on the form of the stem, a number of other changes to the stem may also occur. These are given in Table 3.6. Notable among these changes is that /a/, /i/, and /u/ undergo lengthening when they occur as the final vowel in a stem of the form *-VCCVC₁* where *C₁* is /k/ or /ġ/. The absolutive dual is one of very few Iñupiaq suffixes which trigger vowel lengthening.

TABLE 3.6: Effects of the absolutive dual ending on the preceding stem; data from MacLean (1986a:77–78)

Stem form	Changes caused by absolutive dual
-VCCV, -VV	no changes
-VCV	optional gemination of C
-VVC	deletion of final consonant
-VCCVġ	deletion of /ġ/
-VCVġ	deletion of /ġ/, gemination of C
-VCV ₁ k, -VCV ₁ ġ	deletion of final consonant; if V ₁ is /i/, it becomes [a]
-VCCV ₁ k, -VCCV ₁ ġ	deletion of final consonant; lengthening of V ₁ if it is not /i/; changing of /i/ to [a]

The contents of Table 3.6 are expressed as rules in Figure 3.43 on the next page. They begin with a convenience definition, *AbsDualIgnore*, establishing the symbols which subsequent definitions will ignore. *AbsolutiveDualVowel* handles vowel-final stems (including stems whose absolutive singular form ends in *-n* or *-ñ*), replacing the ABSDUAL symbol with a trigger for optional gemination¹⁹ followed by (k). *AbsolutiveDualCCVStrongC* handles vowel lengthening for cases *-VCCV₁k* and *-VCCV₁ġ*. Finally, the operation *AbsolutiveDualConsonant* arranges for gemination to occur (if applicable) and for the stem-final consonant to be deleted (see page 133) and appends (k) to the stem.

3.8.2.10 Handling Optional Gemination and Stems with “Historic Consonants”

This rule, which I call “pre-gemination,” groups together two operations related to gemination, but which must be evaluated before the gemination rule. The first operation handles the OPTGEM symbol by

¹⁹Note that there is no harm in GEM or OPTGEM following stems where gemination is not possible, such as stems ending in VCCV or VCVV; the gemination rule takes the stem form into account (see Section 3.8.2.10 on pages 124–126 and Section 3.8.2.14 on pages 129–130).

```

define AbsDualIgnore [ \[ Graphemes | ABSDUAL ] ];
define AbsoluteDualVowel [ ABSDUAL -> OPTGEM k || Vowel / AbsDualIgnore _ ];
define AbsoluteDualCCVStrongC [ a -> a a, i -> i i, u -> u u, e -> a ||
  ConsonantCluster _ [[ k | q ] ABSDUAL] / AbsDualIgnore ];
define AbsoluteDualConsonant [ ABSDUAL -> GEM MINUS k ];
define LexiconAbsoluteDual [
  Lexicon .o.
  AbsoluteDualVowel .o.
  AbsoluteDualCCVStrongC .o.
  AbsoluteDualConsonant ];
define Lexicon LexiconAbsoluteDual;

```

FIGURE 3.43: Code for forming the absolute dual

replacing each string containing this symbol with two strings, one of which contains the GEM symbol in its place, the other of which simply deletes the symbol.

The second operation deals with the HISTORICCONSONANT symbol, which signifies the presence of a consonant in the preceding stem which is realized only when the stem is followed directly by a suffix which triggers gemination (see example (23) and discussion on pages 17-18). Stems with “historic consonants,” such as *ku^huk* ‘river’ are entered in data/stems.txt with the historic consonant in place and followed by the HISTORICCONSONANT symbol, for example, *kurukHISTORICCONSONANT*. *KeepHistoricConsonant*, shown in Figure 3.44, retains a historic consonant if the stem where it is found is followed by the GEM sign, signifying that gemination will occur; this rule also deletes the HISTORICCONSONANT symbol in that case. Any strings which still contain the HISTORICCONSONANT symbol after the application of *KeepHistoricConsonant* must therefore not contain the necessary environment for the historic consonant to manifest itself in the surface form, so *RemoveHistoricConsonant* deletes this consonant (the HISTORICCONSONANT symbol in these strings is removed by the code [HISTORICCONSONANT -> 0] in the cascade that follows).

```

define OptionalGemination [ OPTGEM -> [ GEM | 0 ] ];
define KeepHistoricConsonant [ HISTORICCONSONANT -> 0 || Consonant Vowel (Consonant) _ GEM /
  \[Graphemes | GEM] ];
define RemoveHistoricConsonant [ Consonant -> 0 || _ [Vowel (Consonant) HISTORICCONSONANT] /
  \[Graphemes | HISTORICCONSONANT] ];
define LexiconPreGemination [
  Lexicon .o.
  OptionalGemination .o.
  KeepHistoricConsonant .o.
  RemoveHistoricConsonant .o.
  [HISTORICCONSONANT -> 0]];
define Lexicon LexiconPreGemination;

```

FIGURE 3.44: Code for handling optional gemination and “historic” consonants

The reason for separating these operations (the handling of OPTGEM in particular) from the gemination rule has to do with the allomorph conditioning environment filters. Gemination should not be applied until

after these filters, but the operation to resolve HISTORICCONSONANT must be completed before the filters are applied, since the outcome of this operation affects the shape of the stem (and therefore, potentially, whether a filter will accept or reject that stem). The resolution of HISTORICCONSONANT depends in turn upon the resolution of the OPTGEM symbol. Thus, OPTGEM and then HISTORICCONSONANT are handled here, in that order; then, the conditioning environment filters are applied (see Section 3.8.2.11); and finally, somewhat later, gemination is applied (see Section 3.8.2.14 on pages 129–130).

3.8.2.11 Applying Conditioning Environment Filters

At this point in the rules file, the allomorph conditioning environment filters are applied to the transducer. As was discussed in Section 3.3, allomorph conditioning filters are xfst code which eliminates from the transducer any strings containing an allomorph suffixed to a stem whose form it is not compatible with. The code is generated by `xml_to_lexc.tcl` (see Section 3.7.5 on pages 113–114) and is loaded by the command source `ipk_condition_xfst.txt`. The conditioning environment filter code contains instructions to compose the filters below the transducer and to redefine the variable `Lexicon` (see page 115).

3.8.2.12 Deriving Demonstrative Stems

Demonstrative adverbs and pronouns differ from other entries in the transducer in that they are entered as full words in citation form rather than as stems (see pages 74–75). For demonstrative adverbs, this is the “interjectional” form (the form with no case ending); for demonstrative pronouns, it is the absolutive singular form. The biggest disadvantage to listing demonstratives as full words is the inelegance and redundancy of having to list two forms which can be derived more or less predictably from a single root. Happily, this drawback is offset by two key advantages: first, a demonstrative in citation form is more easily recognized than an abstract demonstrative stem, so using citation forms should make the transducer more user-friendly; and second, for both demonstrative adverbs and demonstrative pronouns, the citation forms are the most exceptional members of their respective paradigms, and listing them directly in the lexicon relieves me of the burden of writing two sets of rules, one for citation forms and one for all other forms.

In order to generate non-citation forms, however, the citation forms must be reduced to some kind of stem. Table 3.7 on the following page²⁰ illustrates several forms of nine example demonstratives, including the adverbial and pronominal “stems” which the rules described in this section will derive. These differ from the demonstrative “roots” given by MacLean (1986b:47–48), as well as the “relative stems” given in MacLean (n.d.b:ch. 24) for deriving oblique forms of demonstrative pronouns. They are simply greatest common denominators²¹ of the non-citation forms of each demonstrative pronoun or adverb, which will be followed by the endings shown in Figure 3.18 on page 95.

3.8.2.12.1 DEMONSTRATIVE ADVERB STEMS For demonstrative adverbs, identifying the “stem” involves removing the final vowel of the citation form as well as zero, one, or two consonants, depending on the consonants. The demonstrative adverb stem-deriving rule is shown in Figure 3.45 on page 128. This

²⁰Data for this table was drawn from MacLean [1986b:47–48, 122–125], MacLean [n.d.b:ch. 24], and Seiler [2005:461]. The “stem” columns are based on analyses by J. Eliot DeGolia and myself.

²¹For demonstrative pronouns, that’s not entirely true; non-singular demonstrative pronoun endings delete the final consonant of these “stems.”

TABLE 3.7: Examples of demonstratives with adverbial and pronominal citation forms and “stems”

Root	Definition	Adverbial				Pronominal			
		Interj.	Loc.	Via.	Stem	Abs.Sg.	Abs.Pl.	Rel.Sg.	Stem
<i>kîg-</i>	out there nearby, restricted	<i>kigga</i>	<i>kiani</i>	<i>kiuna</i>	<i>kî</i>	<i>kigña</i>	<i>kikkua</i>	<i>kiktuma</i> <i>kiksruma</i> <i>kigruma</i>	<i>kikt</i>
<i>îk-</i>	over there; across there; restricted	<i>ikka</i>	<i>ikani</i>	<i>ikuuna</i>	<i>îk</i>	<i>igña</i>	<i>ikkua</i>	<i>iktuma</i> <i>iksruma</i> <i>igruma</i>	<i>ikt</i>
<i>îm-</i>	there, aforementioned (in past)	<i>imma</i>	<i>imani</i>	<i>imuuna</i>	<i>îm</i>	<i>imña</i>	<i>ipkua</i>	<i>iptuma</i> <i>ivsruma</i> <i>ivruma</i>	<i>ipt</i>
<i>akîm-</i>	across there; not visible	<i>aḡma</i>	<i>aḡmani</i>	<i>aḡmuuna</i>	<i>aḡm</i>	<i>akimna</i>	<i>akipkua</i>	<i>akiptuma</i> <i>akivsruma</i> <i>akivruma</i>	<i>akipt</i>
<i>un-</i>	down there; near or on sea, down the coast; near exit; extended	<i>unna</i>	<i>unani</i>	<i>unuuna</i>	<i>un</i>	<i>unna</i>	<i>utkua</i>	<i>uttuma</i> <i>utsruma</i> <i>urruma</i>	<i>utt</i>
<i>mar-</i>	here; extended	<i>marra</i>	<i>maani</i>	<i>mauna</i>	<i>ma</i>	<i>manna</i>	<i>matkua</i> , <i>makkua</i> , <i>makua</i>	<i>mattuma</i> <i>matsruma</i> <i>marruma</i>	<i>matt</i>
<i>qav-</i>	in there; up the coast, to the east; extended	<i>qavva</i>	<i>qavani</i>	<i>qavuuna</i>	<i>qav</i>	<i>qamna</i>	<i>qapkua</i>	<i>qaptuma</i> <i>qavsruma</i> <i>qavruma</i>	<i>qapt</i>
<i>uv-</i>	here; restricted	<i>uvva</i>	<i>uvani</i>	<i>uvuuna</i>	<i>uv</i>	<i>una</i>	<i>ukua</i>	<i>uuma</i>	<i>u</i>
<i>tat+uv-</i>	there, near listener, away from speaker; restricted	<i>tavra</i> , <i>tarva</i>	<i>tavrani</i> , <i>tarvani</i>	<i>tavrana</i> , <i>tarvana</i>	<i>tavr</i> , <i>tarv</i>				

rule is triggered by a morpheme boundary followed by the multicharacter symbol DASTEM, which begins all non-citation demonstrative adverb endings (see Figure 3.18 on page 95). DaStemTavra is specific to the demonstrative adverb *tavra/tarva* ‘there (close to listener but not speaker)’, deleting the final vowel. DaStemGGA handles all demonstrative adverbs ending in {gga}, deleting all three symbols. DaStemŋMA applies to demonstrative adverbs ending in ḡma and gma and deletes only the final vowel. All other demonstrative adverbs except for *marra* are handled by DaStemDefault, which deletes the final vowel and a single preceding consonant. *Marra* gets its own rule, DaStemRRA, which is written differently from the surrounding rules. The “stem” of *marra* is {ma}, but if this rule were only to delete the substring {rra}, that would leave the string ma>DASTEM, which would be changed to >DASTEM by DaStemDefault—something which obviously should not happen. To prevent this unwanted outcome, DaStemRRA deletes not only the non-stem part of the citation form, but also the rule trigger DASTEM. Because the rule trigger is on the other side of a morpheme boundary, the rule also deletes the morpheme boundary but then restores it (-> %>). An alternative approach would have been to write DaStemRRA following the same pattern as the other operations and then to compose it below DaStemDefault.

The DeleteExtraVowel operation is provided to fix the vialis forms of demonstrative adverbs in cases where the “stem” ends in a vowel (for example, *qigga* [stem {qi}], *marra* [stem {ma}]). Without this rule, which deletes the first vowel of the demonstrative adverbial vialis ending, these demonstratives would

```

define DaStemTavra [ a -> 0 || t a [v r | r v] _ MorphemeBoundary DASTEM ];
define DaStemRRA [ r r a MorphemeBoundary DASTEM -> %> ];
define DaStemGGA [ g g a -> 0 || _ MorphemeBoundary DASTEM ];
define DaStemŋMA [ a -> 0 || [ŋ | g] m _ MorphemeBoundary DASTEM ];
define DaStemDefault [ [v | k | m | n | ñ ] a -> 0 || _ MorphemeBoundary DASTEM ];
define DeleteExtraVowel [
  [ Vowel -> 0 || Vowel MorphemeBoundary DELETEEXTRAVOWEL _ ]];
define LexiconDaStem [
  Lexicon .o.
  DaStemTavra .o.
  DaStemRRA .o.
  DaStemGGA .o.
  DaStemŋMA .o.
  DaStemDefault .o.
  [ DASTEM -> 0 ] .o.
  DeleteExtraVowel .o.
  [ DELETEEXTRAVOWEL -> 0 ]];
define Lexicon LexiconDaStem;

```

FIGURE 3.45: Code for deriving stems from demonstrative adverbs

be generated with illegal three-vowel clusters (e.g., **qiuuna*, **mauuna*). The `DeleteExtraVowel` operation is triggered by the symbol `DELETEEXTRAVOWEL`, which is included in the definition of the demonstrative adverb `vialis` ending (see Figure 3.45).

The cascade which adds the demonstrative adverb stem-deriving rule to the transducer deletes the symbols `DASTEM` and `DELETEEXTRAVOWEL`.

3.8.2.12.2 DEMONSTRATIVE PRONOUN STEMS Demonstrative pronoun stems are derived via the following steps: the final `/na/` is deleted; if the stem contains `/i/`, this is changed to `/i/` (to prevent a following `/t/` from palatalizing to `[s]`, giving forms like **kiksuma* or **ipsuma*); and the last non-deleted consonant (if any) is obstruentized and appended with `/t/` (in the case of *una* ‘this’, there are no non-deleted consonants and `/t/` is not appended, resulting in the stem `{u}`). These steps are implemented in the operations `DpStemRemoveNA`, `DpStemSoftenStrongl`, and `DpStemObstruentize` in Figure 3.46 on the next page. Operations `DpStemOptVoice` and `DpStemOptSpirantize` generate variant forms of singular non-absolute demonstrative pronouns where the stem consonants are voiced or affricated (see relative singular forms in Table 3.7 on the preceding page). The right-hand context for these operations is `MorphemeBoundary DPSTEM \MINUS`, exploiting the fact that singular demonstrative pronoun endings do not delete the final consonant of the stem, while all other endings do; this ensures that voiced and spirantized variants are only generated for singular forms.

As usual, the cascade contains code to remove the rule trigger symbol `DPSTEM`.

3.8.2.13 “Weakening” `/i/` Due to a Following Postbase

The initial `/t/` of postbase *-tualuk* or *-tuaq* ‘the only ____’ is exempt from palatalization. This is handled in the transducer by “weakening” a preceding `/i/` to `/i/` when the rule trigger `WEAKENI` is present. The code for this rule is given in Figure 3.47 on the next page.


```

define DpStemRemoveNA [ [ n | ñ ] a -> 0 || _ MorphemeBoundary DPSTEM ];
define DpStemSoftenStrongl [ i -> e || _ (Consonant) MorphemeBoundary DPSTEM ];
define DpStemObstruentize [ m -> p t, n -> t t, g -> k t || _ MorphemeBoundary DPSTEM ];
define DpStemOptVoice [ t t (->) r r, k t (->) g r, p t (->) v r || _ MorphemeBoundary DPSTEM \MINUS ];
define DpStemOptSpirantize [ t t (->) t s r, k t (->) k s r, p t (->) v s r || _ MorphemeBoundary DPSTEM \MINUS ];
define LexiconDpStem [
  Lexicon .o.
  DpStemRemoveNA .o.
  DpStemSoftenStrongl .o.
  DpStemObstruentize .o.
  DpStemOptVoice .o.
  DpStemOptSpirantize .o.
  [ DPSTEM -> 0 ]];
define Lexicon LexiconDpStem;

```

FIGURE 3.46: Code for deriving stems from demonstrative pronouns

```

define Weakenl [ i -> e || _ (Consonant) WEAKENI / \[ Graphemes | WEAKENI ]];
define LexiconWeakenl [ Lexicon .o. Weakenl .o. [ WEAKENI -> 0 ]];
define Lexicon LexiconWeakenl;

```

FIGURE 3.47: Code for deriving stems from demonstrative pronouns

3.8.2.14 Gemination

The rule discussed in this section implements gemination, which was discussed in Section 2.1.2.5 on pages 16–18. This rule is triggered by the symbol GEM, which is included directly in the definition of allomorphs which require gemination, and inserted by the “pre-gemination” rule into strings containing allomorphs which allow but do not require gemination (see Section 3.8.2.10 on pages 124–126). In addition to the presence of the GEM symbol, stems must be of the form -VCV(C) in order for gemination to occur.

One of the side-effects of gemination is that when /i/ follows the consonant to be geminated, it becomes [a] (see pages 12–14). In the rule shown in Figure 3.48 on the following page, this is accomplished by the operation WeaklToA, which changes ⟨e⟩ to ⟨a⟩ in contexts where gemination will occur.

Gemination is also responsible for the obstruentizing of certain instances of /ç/, /ʁ/, and /j/ and the surfacing of “historic consonants” (see examples (22) and (23) on page 18 and discussion on pages 17–18). “Historic consonants” were taken care of by the “pre-gemination” rule described in Section 3.8.2.10 on pages 124–126. The obstruentizing of /ç/ /ʁ/ is handled by the operation GeminateHistoricStops, which is triggered by the symbol HISTORICSTOP, while the gemination of /j/ to /tʃ/ is accomplished via the operation GeminateYTCH which is sensitive to the symbol YTCH. In retrospect, GeminateHistoricStops and GeminateYTCH could have been collapsed into a single rule with a single rule trigger, but I believe the fact that the two operations stem from separate historical changes justifies the current implementation.

The convenience definition GeminationRightContext specifies what must follow a consonant (including which symbols may be ignored) in order for that consonant to undergo gemination. This code could have been included directly in the rule GeminateOthers (which was originally a series of rules, one for each

```

define WeakIToA [ e -> a || Vowel Consonant _ [(Consonant) GEM] / [\[Graphemes | GEM]] ];
define GeminateHistoricStop [ g -> k k, ġ -> q q || Vowel _ [Vowel (Consonant) HISTORICSTOP GEM]
  / [\[Graphemes | GEM | HISTORICSTOP]] ];
define GeminateYTCH [ y -> t c h || Vowel _ [Vowel (Consonant) YTCH GEM] / [\[Graphemes | GEM | YTCH]] ];
define GeminationRightContext [ Vowel (Consonant) GEM ] / [\[Graphemes | GEM]]; ! convenience definition
define GeminateOthers [ p -> p p, t -> t t, k -> k k, AnyQ -> q q, v -> v v, l -> l l, ! -> ! !, y -> y y, r -> r r,
  m -> m m, n -> n n, ñ -> ñ ñ, ŋ -> ŋ ŋ, s -> t c h, g -> g g, ġ -> ġ ġ
  || Vowel _ GeminationRightContext ];
define LexiconGemination [
  Lexicon .o.
  WeakIToA .o.
  GeminateHistoricStop .o.
  [ HISTORICSTOP -> 0 ] .o.
  GeminateYTCH .o.
  [ YTCH -> 0 ] .o.
  GeminateOthers .o.
  [ GEM -> 0 ];
define Lexicon LexiconGemination;

```

FIGURE 3.48: Code to implement gemination

consonant to be geminated), but I find that keeping the two separated makes each easier to read.

With previous rules having handled all unusual cases of gemination, the operation `GeminateOthers` takes care of all regular gemination. The cascade that follows deletes rule trigger symbols `HISTORICSTOP`, `YTCH`, and `GEM`.

3.8.2.15 Velarization of Stem-Final /q/

Enclitics which begin with /k/, such as =*kiaq* ‘I wonder; I think’ and =*kii* ‘because’, cause a preceding /q/ to become [k]. The code to accomplish this is shown in Figure 3.49. In its current implementation, the rule is sensitive to the trigger symbol `VELARIZE`, which is specified as the first symbol for all enclitics beginning with ⟨k⟩; however, as the transducer does not contain any enclitics which are exceptions to this rule, it may make sense to rewrite the rule to be sensitive to an enclitic boundary and an enclitic-initial ⟨k⟩ instead.

```

define LexiconVelarize [
  Lexicon .o.
  [ AnyQ -> [ 0 | k ] || _ VELARIZE / [\[ Graphemes | VELARIZE ] ] ] .o.
  [ VELARIZE -> 0 ];
define Lexicon LexiconVelarize;

```

FIGURE 3.49: Code to implement velarization of stem-final /q/

3.8.2.16 Uvularization of Enclitic-Initial /g/

Some speakers uvularize the initial consonant of the enclitic =*gguuq* ‘it is said’ when the word to which it is attached ends in /q/:

- Ex. (68) a. *puktaġġuuquna*
puktaaġ=gguuq=una
 iceberg.ABS.SG=it.is.said=this
 ‘The iceberg, it is said ...’ (Nageak 1975:25)
- b. *ikiġġaaqpaqaqsimaruġġuuq*
ikiġġaat-qpak-qaq-sima-ruaq=gguuq
 platform.cache-big-to.have-it.is.now.known-ind.pst.3sg=it.is.said
 ‘He did have a big platform cache, it is said’ (Ahvakana 1973:18)

The rule implementing this change is very simple, as can be seen in Figure 3.50. It is triggered by the symbol UVULARIZE and a uvular preceding the enclitic. As =*gguuq* is the only formative in the lexicon to contain the rule trigger, it is the only formative to be affected by this rule.

```
define LexiconUvularize [
  Lexicon .o.
  [ g -> ġ || [ [ AnyQ | ġ ] UVULARIZE ] / [ \ Graphemes | UVULARIZE ] ] _ .o.
  [ UVULARIZE -> 0 ] ;
define Lexicon LexiconUvularize;
```

FIGURE 3.50: Code to implement uvularization of enclitic-initial /g/

3.8.2.17 Vowel Lengthening

Vowel lengthening is a rare phenomenon in Iñupiaq. I am aware of only three suffixes which trigger it: the absolute dual (see Section 3.8.2.9 on pages 123–124), the “vocative possessive” suffix -*Vŋ* ‘my dear ____’, and the postbase -*Vq*- ‘to say ____; to be or play ____’. Vowel lengthening triggered by the absolute dual was handled previously. The rule shown in Figure 3.51 on the following page handles the other two cases. It is triggered by the symbol LENGTHENV and only applies when the preceding stem ends in -CV(C). LengthenVDeleteConsonant deletes any stem final consonant; then LengthenV lengthens the stem final vowel unless it is /i/.

3.8.2.18 Handling Allomorphy Shorthand Notations

As discussed on page 80, the Iñupiaq transducer borrows from MacLean’s Iñupiaq reference works two shorthand notations for common allomorphy patterns. One of these is brackets around an initial consonant, signifying that that consonant is present when the preceding stem ends in a vowel or /t/, and

```

define LengthenVRightContext [LENGTHENV / [\ Graphemes | LENGTHENV ] ]; ! convenience definition
define LengthenVDeleteConsonant [ Consonant -> 0 || _ LengthenVRightContext ];
define LengthenV [ a -> a a, i -> i i, u -> u u || Consonant _ LengthenVRightContext ];
define LexiconLengthenV [
  Lexicon .o.
  LengthenVDeleteConsonant .o.
  LengthenV .o.
  [ LENGTHENV -> 0 ];
define Lexicon LexiconLengthenV;

```

FIGURE 3.51: Code to lengthen vowels

absent otherwise. The rule which handles this notation is given in Figure 3.52. It consists of three steps: if the left bracket is preceded by a vowel or (t), then the left bracket is deleted; any remaining left brackets are deleted together with whatever symbol follows; and finally, the right bracket is deleted.

```

define LexiconBrackets [
  Lexicon .o.
  [ %[ -> 0 || [ Vowel | t ] / [\Graphemes | %[]] _ Consonant % ] ] .o.
  [ %[ ? -> 0 ] ] .o.
  [ %[ -> 0 ] ];
define Lexicon LexiconBrackets;

```

FIGURE 3.52: Code for handling bracket notation

The other shorthand notation is to stack one consonant over another; this has been implemented in the transducer by listing the top consonant followed by a double forward slash // followed by the bottom consonant. This notation indicates that the top or first consonant is to be used following a consonant, and the bottom consonant is to be used following a vowel. The rule which makes this happen is shown in Figure 3.53. If the // symbol is preceded by two consonants, then the symbol and the following consonant are deleted; any // symbols left in the lower language of the transducer are then deleted together with the preceding consonant.

```

define LexiconConsonantOverConsonant [
  Lexicon .o.
  [ "//" Consonant -> 0 || Consonant Consonant / [\Graphemes | "//"] _ ] .o.
  [ Consonant "//" -> 0 ] ];
define Lexicon LexiconConsonantOverConsonant;

```

FIGURE 3.53: Code for handling consonant over consonant notation

3.8.2.19 Suffix Attachment Patterns

Section 2.1.2.6 on pages 18-22 discussed MacLean’s system for coding suffix attachment patterns. This system is used within the transducer; the symbols used to denote these patterns in the lexical data files are given in Table 3.3 on page 79, and those used in the xfst/lexc files are given in Table 3.5 on page 111. This last set of symbols serve as rule triggers for the rules to be discussed in this section, one for each of the following patterns (named after the symbols used by MacLean): minus, plus, colon, division, plus over minus, minus over plus, equals, and tilde.

3.8.2.19.1 **MINUS PATTERN** The minus pattern is one of the simplest suffix attachment patterns. It simply deletes any stem-final consonant. The code for this rule is given in Figure 3.54.

```
define LexiconMinus [
  Lexicon .o.
  [ Consonant -> 0 || _ MINUS / [ \ [ Graphemes | MINUS ] ] ] .o.
  [ MINUS -> 0 ] ];
define Lexicon LexiconMinus;
```

FIGURE 3.54: Code implementing the “minus” suffix attachment pattern

3.8.2.19.2 **PLUS PATTERN** Perhaps counterintuitively, the plus pattern is slightly more complex than the preceding pattern. (Code for this rule is shown in Figure 3.55.) Although theoretically this pattern simply calls for the suffix to be appended with no deletions to the stem (which requires no rule at all), the initial consonant of the suffix will be deleted in cases where the suffix begins with a consonant cluster and attaches to a consonant-final stem; this is handled by the operation PlusDeleteConsonant. Additionally, a stem-final consonant will undergo assimilation with the initial segment of the suffix (specifically, a stem-final back consonant followed by a suffix-initial voiced phoneme will become a voiced fricative; this is accomplished by PlusFricativizeBackStop). There is some overlap between this assimilation operation and the assimilation rule (see Section 3.8.2.26 on pages 141-142); this limited amount of assimilation is included in this rule for the case where a suffix begins with a vowel, since the assimilation rules deal only with consonant clusters.

```
define PlusDeleteConsonant [ Consonant -> 0 || [ Consonant PLUS ] / [ \ [ Graphemes | PLUS ] ] _ Consonant ];
define PlusFricativizeBackStop [ k -> g, AnyQ -> ġ || _ [ PLUS VoicedPhoneme ] / [ \ [ Graphemes | PLUS ] ] ];
define LexiconPlus [
  Lexicon .o.
  PlusDeleteConsonant .o.
  PlusFricativizeBackStop .o.
  [ PLUS -> 0 ] ];
define Lexicon LexiconPlus;
```

FIGURE 3.55: Code implementing the “plus” suffix attachment pattern

3.8.2.19.3 **COLON PATTERN** This complicated pattern is described in Table 2.4 on page 19. To recapitulate, this rule changes the stem in one of four ways, depending on the form of the stem:

- for stems ending in $-VCiC$, /i/ is deleted and the second consonant assimilates to agree with the first in terms of voicing, continuancy, and optionally, nasality
- for stems ending in $-VCCiC$, two outcomes are possible: either the final consonant becomes a voiced fricative, or /i/ is treated as /i/ and the $-VC$ pattern below applies
- for stems ending in $-VC$, where V is not /i/, the final consonant is deleted
- for stems ending in $-V$, the stem is attached directly (stem-final /i/ becomes [a]; potential three-vowel clusters are broken up by /ʃ/)

The implementation of this rule, given in Figure 3.56, consists of five steps (plus convenience definition `ColonIgnoreSymbols`), all triggered by the symbol `COLON`. `ColonOptStrengthenE` optionally changes ⟨e⟩ (representing /i/) to ⟨i⟩ (representing /i/) in the preceding stem if it ends in $-CCiC$. `ColonDeleteFinalConsonant` deletes the stem-final consonant if it is preceded by a vowel other than /i//e/. `ColonDeleteWeakI` deletes /i//e/ in stems ending in $-VCiC$. `ColonOptNasalizeK` optionally changes stem-final ⟨k⟩ to ⟨ŋ⟩ following a nasal. `ColonFricativizeBackStop` changes stem-final ⟨k⟩ to ⟨g⟩ and ⟨q⟩ to ⟨ǰ⟩ following a voiced phoneme. The operations which may pertain to each of the four stem patterns are listed in Table 3.8 on the following page.

```

define ColonIgnoreSymbols [Graphemes | COLON]; ! convenience definition
define ColonOptStrengthenE [ [e | é] (->) i || ConsonantCluster _ [ Consonant COLON ] / ColonIgnoreSymbols ];
define ColonDeleteFinalConsonant [ Consonant -> 0 || [ Vowel - [e | é] ] _ COLON / ColonIgnoreSymbols ];
define ColonDeleteWeakI [ [ e | é ] -> 0 || Vowel Consonant _ [ Consonant COLON ] / ColonIgnoreSymbols ];
define ColonOptNasalizeK [ k (->) ŋ || Vowel Nasal _ COLON / ColonIgnoreSymbols ];
define ColonFricativizeBackStop [ k -> g, AnyQ -> ǰ || VoicedPhoneme _ COLON / ColonIgnoreSymbols ];
define LexiconColon [
  Lexicon .o.
  ColonOptStrengthenE .o.
  ColonDeleteFinalConsonant .o.
  ColonDeleteWeakI .o.
  ColonOptNasalizeK .o.
  ColonFricativizeBackStop .o.
  [ COLON -> 0 ] ];
define Lexicon LexiconColon;

```

FIGURE 3.56: Code implementing the “colon” suffix attachment pattern

3.8.2.19.4 **DIVISION PATTERN** The division pattern deletes stem-final /ǰ/. The code is given in Figure 3.57 on the next page (recall that ⟨q⟩ represents /ǰ/ within the transducer and that ⟨q̇⟩ represents /ǰ̇/). It is sensitive to the symbol `DIVISION`.

3.8.2.19.5 **PLUS OVER MINUS PATTERN** This pattern deletes a stem-final back consonant; it will also delete the initial consonant of the suffix if necessary to prevent a three-consonant cluster. Figure 3.58 on the following page contains the code which implements this rule. This rule is triggered by the symbol `PLUSOVERMINUS`.

TABLE 3.8: Stem patterns and the “colon” rule operations which may affect them

Pattern	Operations which may apply
-VCiC	ColonDeleteWeakI, ColonOptNasalizeK, ColonFricativizeBackStop
-VCCiC	ColonOptStrengthenE (in which case the operations on the next row apply instead of the other operations on this row), ColonFricativizeBackStop
-VC (V ≠ /i/)	ColonDeleteFinalConsonant
-V	not affected by any “colon” rule operation

```

define LexiconDivision [
  Lexicon .o.
  [ q -> 0 || _ DIVISION / [ \ Graphemes | DIVISION ] ] .o.
  [ DIVISION -> 0 ] ];
define Lexicon LexiconDivision;

```

FIGURE 3.57: Code implementing the “division” suffix attachment pattern

```

define LexiconPlusOverMinus [
  Lexicon .o.
  [ [ k | AnyQ ] -> 0 || _ PLUSOVERMINUS / [ \ Graphemes | PLUSOVERMINUS ] ] .o.
  [ Consonant -> 0 || [ Consonant PLUSOVERMINUS ] / [ \ Graphemes | PLUSOVERMINUS ] _ Consonant ] .o.
  [ PLUSOVERMINUS -> 0 ] ];
define Lexicon LexiconPlusOverMinus;

```

FIGURE 3.58: Code implementing the “plus over minus” suffix attachment pattern

3.8.2.19.6 MINUS OVER PLUS PATTERN Minus over plus deletes a stem-final /t/. The code for this rule is shown in Figure 3.59. The rule trigger for this rule is MINUSOVERPLUS.

```

define LexiconMinusOverPlus [
  Lexicon .o.
  [ t -> 0 || _ MINUSOVERPLUS / [ \ Graphemes | MINUSOVERPLUS ] ] .o.
  [ MINUSOVERPLUS -> 0 ] ];
define Lexicon LexiconMinusOverPlus;

```

FIGURE 3.59: Code implementing the “minus over plus” suffix attachment pattern

3.8.2.19.7 EQUALS PATTERN This pattern and the tilde pattern are less common patterns, attested mostly in postbases which today are marginally productive at best. It’s probably for this reason that they receive

less thorough treatments than the patterns discussed above. At present, only three allomorphs in the transducer are specified with the equals pattern (definitions from MacLean n.d.a):

- *-îlaq(-)* ‘one without, area which lacks ____; one who is not ____; to be or do without ____’
- *-îqî-* (variant of *-lîqî-*) ‘having to do with hunting, preparing, repairing, cleaning, associating, messing around with ____’
- *-uraq* ‘a smaller version of a ____; a diminutive ____; immediate vicinity of ____’

MacLean (n.d.a) does not specify any conditioning environments for the “equals” variants of *-îlaq(-)* and *-lîqî-*, but she does indicate that the productivity of *-îlaq(-)* is limited, and she notes in the definition of *-lîqî-* that “the meaning of the long form *-lîqî-* differs somewhat from the meaning of the short form ‘=*lîqî-*; the words containing the short form are more specific, often lexicalized.” The variant of *-uraq* which exhibits the equals pattern is restricted to stems ending in *-VVC*, but MacLean gives no indication that the productivity of this allomorph is limited.

MacLean (1986b:102) describes the equals pattern as follows: “the equals sign ...is used to indicate that both the semi-final vowel and the final consonant are deleted before the addition of the postbase.”

Figure 3.60 details the implementation of the equals pattern. The rule trigger for this rule is the symbol EQUAL. The first operation in the rule is a filter to disallow the equals pattern from operating on monosyllabic consonant-final stems. This represents a bit of speculation on my part, but it prevents the transducer from generating a number of very unlikely candidate words. After filtering, the rule deletes the final vowel and consonant of the stem,²² and finally the EQUAL symbol.

```
define LexiconEqual [
  Lexicon .o.
  [ ~[ [ .#. (Consonant) Vowel (Vowel) Consonant EQUAL ?* ] / [ \ Graphemes | EQUAL ] ] ] .o.
  [ Vowel Consonant -> 0 || _ EQUAL / [ \ Graphemes | EQUAL ] ] ] .o.
  [ EQUAL -> 0 ] ];
define Lexicon LexiconEqual;
```

FIGURE 3.60: Code implementing the “equals” suffix attachment pattern

3.8.2.19.8 TILDE PATTERN This obscurest of patterns involves deletion of stem-final /i/ and, if the suffix begins with /î/, regressive palatalization of a preceding /t/ to [s] (see MacLean 1986b:105). Few if any productive postbases exhibit this pattern; the Iñupiaq transducer lexicon currently includes only one: *-qsraq-* ‘to continue to ____; to experience ____.’ The rule is based as much on examples from the definition of this morpheme in MacLean (n.d.a) as from any description of the pattern itself; several of these examples are reproduced in (69) and (70) below. Since the tilde pattern applies only to stems ending in /i/ (Edna MacLean, personal communication, 20 October 2009), only examples with /i/-final stems are given.

²²This means the stem must end in a consonant to be affected by this rule; for vowel-final stems consisting of more than one syllable, this rule will be equivalent to the plus pattern rule. This may be wrong; it may well be possible that with vowel-final stems, the final vowel is deleted, or it may be the case that allomorphs exhibiting this pattern should not be allowed after a vowel-final stem.

- Ex. (69) a. *anmiksraqtuq*
anmigĩ-qsraq-tuq
 feel.onself.more.capable.than-continue.to-IND.PRS.3SG
 ‘he is acting superior’
- b. *piᅇiksraqtuq*
piᅇigĩ-qsraq-tuq
 worry.about-experience-IND.PRS.3SG
 ‘she is worrying about someone’s safety’
- Ex. (70) a. *agliqsraqtuq*
agligĩ-qsraq-tuq
 shun.due.to.taboo-continue.to-IND.PRS.3SG
 ‘he is avoiding someone who has a disease’
- b. *anniqsraqtuq*
annigĩ-qsraq-tuq
 refuse.to.share-continue.to-ind.prs.3sg
 ‘she is not willing to share for fear that it might be damaged’

These examples suggest that if an allomorph exhibiting the tilde pattern begins with a consonant cluster, the first consonant of the allomorph is deleted, and, as is generally true with consonant clusters in Iñupiaq, the first consonant assimilates to agree with the second following the rules described in 2.1.2.1.

The implementation of the tilde pattern is reproduced in Figure 3.61 on the next page. Like the equals pattern, it begins with some filters; the first prevents tilde-pattern allomorphs from following stems that do not end in /i/, while the second prevents such allomorphs from following monosyllabic stems. Next, /i/ ((e) or (é)) is deleted if the following symbol is TILDE. If this results in a cluster of three (or more) consonants, the first consonant following the TILDE symbol is deleted (I’m not sure what should happen if the suffix begins with two consonants and the stem ends in two consonants and /i/). Next, (ǵ) and (g) are changed to (q) and (k), respectively, if the consonant after the TILDE symbol is a voiceless fricative. Two comments about this operation: first, it was written specifically with *-qsraq-* in mind, and should additional tilde-pattern formatives be added to the lexicon, additional assimilation rules might need to be added; second, one might one might argue that this operation belongs in the assimilation rule, but because underlying consonant clusters such as (ǵsr) are uncommon, I have elected to put the operation in the only rule where I believe such clusters can possibly arise. The final step in this rule is to replace the symbol TILDE with REGRESSIVEPALATALIZATION so that the regressive palatalization rule (see Section 3.8.2.20 below) will fire if applicable. This operation also serves to remove the symbol TILDE from the lower language of the transducer.

3.8.2.20 Regressive Palatalization

Certain postbases trigger regressive palatalization—the changing of a preceding /t/ to [s] when the postbase begins with /i/ and has caused the deletion of a vowel and possibly a consonant between the /t/ and the /i/. At present, the only formative in the Iñupiaq transducer specified as triggering regressive

```

define LexiconTilde [
  Lexicon .o.
  [ ~$[ [[Graphemes - [e | é]] TILDE ] / [ \Graphemes | TILDE ] ] ] .o. ! disallow pattern with stems that don't end in weak i
  [ ~[ [.#. (Consonant) [e | é] TILDE ?*] / [ \Graphemes | TILDE ] ] ] .o. ! disallow pattern with stems that are too short
  [ [ e | é ] -> 0 || _ TILDE / [ \Graphemes | TILDE ] ] .o.
  [ Consonant -> 0 || [ Consonant TILDE ] / [ \Graphemes | TILDE ] ] _ Consonant ] .o.
  [ ġ -> q, g -> k || _ [ TILDE VoicelessFricative ] / [ \Graphemes | TILDE ] ] .o.
  [ TILDE -> REGRESSIVEPALATALIZATION ] ];
define Lexicon LexiconTilde;

```

FIGURE 3.61: Code implementing the “tilde” suffix attachment pattern

palatalization is *-îqî-* (variant of *-îqî-*) ‘having to do with hunting, preparing, repairing, cleaning, associating, messing around with ____’. MacLean (1986b:105) and MacLean (n.d.a) lead me to believe that allomorphs which combine with the tilde pattern and begin with */î/* also trigger regressive palatalization, and therefore I have written the trigger for this rule into the tilde pattern rule (see pages 136–137).

Regressive palatalization is handled by the rule shown in Figure 3.62. It is triggered by the symbol REGRESSIVEPALATALIZATION and simply changes ⟨t⟩ to ⟨s⟩ when followed by the rule trigger and ⟨i⟩, then removes the rule trigger from the lower language of the transducer.

```

define LexiconRegressivePalatalization [
  Lexicon .o.
  [ t -> s || _ [ REGRESSIVEPALATALIZATION i ] / [ \Graphemes | REGRESSIVEPALATALIZATION ] ] ] .o.
  [ REGRESSIVEPALATALIZATION -> 0 ] ];
define Lexicon LexiconRegressivePalatalization;

```

FIGURE 3.62: Code implementing the “tilde” suffix attachment pattern

3.8.2.21 Preventing */i/* from Becoming [a]

When postbase *-anik-* ‘to have ____ previously, already’ attaches to a stem ending in */î/*, the */î/* does not become [a] but remains [i] (MacLean n.d.a). This is accomplished in the transducer by the rule trigger PROTECTWEAKI in conjunction with the rule given in Figure 3.63. This rule simply changes ⟨e⟩ to ⟨é⟩ when followed by PROTECTWEAKI, since only unaccented ⟨e⟩ is modified by the rule which prevents */î/* in vowel clusters (see Section 3.8.2.27 on page 142).

```

define LexiconProtectWeakI [
  Lexicon .o.
  [ e -> é || _ PROTECTWEAKI / [ \Graphemes | PROTECTWEAKI ] ] ] .o.
  [ PROTECTWEAKI -> 0 ] ];
define Lexicon LexiconProtectWeakI;

```

FIGURE 3.63: Code to prevent */î/* from becoming [a]

3.8.2.22 Breaking Up Three-Vowel Clusters

Iñupiaq does not allow clusters of more than two vowels (see pages 14–16 and Section 2.1.3.2 on page 25). When such clusters would be created by possessive noun endings, they are broken up by /ŋ/; in other cases, they are broken up by /ɣ/. In the transducer, this is handled by the rule shown in Figure 3.64. This rule replaces ⟨a⟩, ⟨i⟩, or ⟨u⟩ with ⟨ga⟩, ⟨gi⟩, and ⟨gu⟩, respectively, when preceded by either a word boundary and two vowels or a consonant and two vowels. The specification of a preceding consonant or word boundary is probably an unnecessary precaution (it would ensure that four-vowel clusters are dealt with appropriately, but I do not believe that four-vowel clusters are possible, even underlyingly).

```
define LexiconFixThreeVowelClusters [
  Lexicon .o.
  [ a -> g a, i -> g i, u -> g u || [Consonant | .#.][Vowel Vowel] / \Graphemes _ ];
define Lexicon LexiconFixThreeVowelClusters;
```

FIGURE 3.64: Code to break up three-vowel clusters

Originally, this rule used the xfst mechanism for true epenthesis, as shown in Figure 3.65. This operation was simpler but unfortunately did not work. It ignores non-alphabetic symbols in both the left and right contexts, and if such a symbol (in practice the most common such symbol was a morpheme boundary marker) was present between the second and third vowels of a three-vowel cluster, the expression shown in Figure 3.65 triggers double epenthesis: once to the left of the non-alphabetic symbol, and again to the right of it. Rewriting the rule as shown in Figure 3.64 ensured that the ⟨g⟩ was consistently inserted only to the right of any intervening non-alphabetic symbols, because the vowel which the ⟨g⟩ “replaces” is on the right of those symbols.

```
[ [..]-> g || [Consonant | .#.][Vowel Vowel] / \Graphemes _ Vowel / \Graphemes ]
```

FIGURE 3.65: Older code to break up three-vowel clusters (flawed)

3.8.2.23 Optional Deletion of Laterals in Contemporative 1 Endings

The contemporative 1 is a dependent verb mood expressing the manner, means, rationale, or cause of the state or event expressed in the matrix clause (MacLean 1986b:1). It has two aspects, termed “realized” and “unrealized.” With the exception of the first person singular ending, contemporative 1 endings preceded by a back consonant have two variant forms. For endings in the realized aspect, one of these forms begins with *-lu*; the other begins with *-u* and triggers voiceless fricativization of the preceding consonant. For endings in the unrealized aspect, one of these forms begins with *-lu*, and the other begins with *-u* and triggers voiced fricativization of the preceding consonant. Rather than specify two separate

allomorphs for each of these endings, I have opted to specify a single form for each ending, tagged with the rule trigger OPTDELETELATERAL, and to create the rule given in Figure 3.66. This rule has one operation for the realized aspect endings and another for the unrealized aspect endings. Bear in mind that the suffix attachment pattern rules have left their mark on the transducer at this point (see Section 3.8.2.19 on pages 133–137), and consequently, back consonants preceding the unrealized aspect endings will have been assimilated to ⟨g⟩ and ⟨ǵ⟩. The context of the second operation takes this into account.

```
define LexiconOptionalLateralDeletion [
  Lexicon .o.
  [† (->) h || [K | AnyQ] / \Graphemes _ OPTDELETELATERAL / [\Graphemes | OPTDELETELATERAL]] ] .o.
  [! (->) 0 || [g | ǵ] / \Graphemes _ OPTDELETELATERAL / [\Graphemes | OPTDELETELATERAL]] ] .o.
  [OPTDELETELATERAL -> 0] ];
define Lexicon LexiconOptionalLateralDeletion;
```

FIGURE 3.66: Code to optionally delete laterals in certain contemporary 1 endings

3.8.2.24 Palatalization

Palatalization, discussed in Section 2.1.2.2 on pages 9–11, is triggered by /i/ and affects phonemically alveolar consonants in the following consonant group (that is, a single consonant or a consonant cluster). Figure 3.67 shows how this process is implemented in the transducer. In its current implementation, palatalization is treated as optional. This is because many speakers, particularly younger ones, palatalize inconsistently or not at all (Lawrence Kaplan, personal communication, September 10, 2008). For words in the development corpus used for this project (see Section 4.1 on page 146), roughly 0.5% of all types and about 0.3% of all tokens are not recognized if palatalization is mandatory; it must be borne in mind, however, that some or all of these may be typographical or performance errors rather than evidence of genuine language change. Treating palatalization as optional is not an optimal solution, particularly because it will generate many forms that would not be appropriate in situations where prescriptivism is called for. In the future, this might be addressed by creating separate “prescriptive” and “descriptive” (or at least permissive) versions of the transducer.

```
define PalatalizeLŁN [ ! (->) |, † (->) l, n (->) ñ || [i (Consonant)] / \Graphemes _ ];
define OptPalatalizeT [ t (->) c h || i / \Graphemes _ WordBoundary / \Graphemes ];
define PalatalizeTtoS [ t (->) s || [i (Consonant)] / \Graphemes _ Vowel / \Graphemes ];
define LexiconPalatalization [
  Lexicon .o.
  PalatalizeLŁN .o.
  OptPalatalizeT .o.
  PalatalizeTtoS];
define Lexicon LexiconPalatalization;
```

FIGURE 3.67: Code to (optionally) palatalize alveolar consonants following /i/

Aside from its being optional, the implementation of palatalization is quite straightforward. The operation `PalatalizeLkN` palatalizes ⟨l⟩, ⟨l̥⟩, and ⟨n⟩. `OptPalatalizeT`, which would be optional even if the other operations were not, changes ⟨t⟩ to ⟨ch⟩ when it is preceded by /i/ and is the last phoneme in the morphosyntactic word (note that the rule is sensitive to a word boundary, which means that word/enclitic boundary marking must be retained in the transducer’s lower language at until this rule has been composed with the transducer). `PalatalizeTtoS` takes care of assibilation, changing ⟨t⟩ to ⟨s⟩ in the context /i(C)_V/. Absent from the rule is the case of /t/ at the beginning of a consonant cluster; this is because there is no graphemic change in this case, even though there is a phonetic change.

3.8.2.25 Removing Morpheme Boundaries

At this point, all rules which are sensitive to any type of morpheme boundary have been added to the transducer; the rule shown in Figure 3.68 removes them.

```
define LexiconRemoveBoundaries [
  Lexicon .o.
  [ MorphemeBoundary2 -> 0 ] ];
define Lexicon LexiconRemoveBoundaries;
```

FIGURE 3.68: Code to remove morpheme boundary symbols from the lower language of the transducer

3.8.2.26 Consonant Assimilation

Regressive consonant assimilation was discussed in Section 2.1.2.1 on pages 7-8. This phenomenon is probably best described in reference to distinctive features, as with Kaplan’s rule given in Figure 2.2 on page 8. Unfortunately, the transducer is implemented in terms of graphemes rather than features, and indeed for most rules a feature-based implementation would probably be overkill. For this rule, then, we are left to generate a long list of “underlying” consonant clusters and the changes they must undergo to comply with the requirements of assimilation. This can be seen in Figure 3.69 on the following page. Not all conceivable combinations of Iñupiaq consonants have been listed in this rule, only those combinations which I believe may arise from the concatenation of morphemes and the subsequent rules which may operate on those consonants. I also have not listed consonant clusters which already satisfy assimilation constraints. Some of the more unusual underlying clusters which may in theory arise due to the application of the equals or tilde patterns (see pages 135-137) have also been omitted; I would prefer for these to be added to the equals or tilde rules directly as needed. Some other cases of assimilation have been handled by previous rules: both the plus pattern (see pages 133-134) and the colon pattern (see page 134) change stem-final back stops to voiceless fricatives when the following phoneme is voiced. These operations overlap with this rule when the suffix begins with a consonant, but are included for cases where the suffix begins with a vowel. Finally, two operations which are not regressive assimilation have been thrown into this rule for want of a better location; these change underlying ⟨ty⟩ and ⟨ts⟩ to surface ⟨tch⟩. The ⟨ts⟩ →

(tch) operation includes an unusual right context: [Graphemes - r] / Ignore. This is to prevent the sequence (tsr) from becoming (tchr).

```

define Ignore \Graphemes; ! convenience definition
define LexiconAssimilation [
  Lexicon .o.
  [ AnyQ -> ġ || _ [Nasal | l | !] / Ignore ] .o.
  [ [ k | g ] (->) ŋ || _ Nasal / Ignore ] .o.
  [ k -> g || _ [Nasal | l | !] / Ignore ] .o.
  [ t -> ñ || _ ñ / Ignore ] .o.
  [ t -> n || _ Nasal / Ignore ] .o.
  [ t -> l || _ l / Ignore ] .o.
  [ t -> ʎ || _ ʎ / Ignore ] .o.
  [ t -> ! || _ ! / Ignore ] .o.
  [ t -> ʀ || _ ʀ / Ignore ] .o.
  [ t -> r || _ r / Ignore ] .o.
  [ t -> r || _ v / Ignore ] .o.
  [ t -> r || _ g / Ignore ] .o.
  [ t -> y || i / Ignore _ g / Ignore ] .o. ! may need to revisit this rule
  [ c h (->) l || _ l / Ignore ] .o. ! would only apply before enclitics
  [ m (->) v || _ l / Ignore ] .o.
  [ n (->) l || _ l / Ignore ] .o.
  [ p -> v || _ [VoicelessFricative | VoicedFricative] / Ignore ] .o.
  [ y -> c h || t / Ignore _ ] .o. ! not properly assimilation
  [ s -> c h || t / Ignore _ [Graphemes - r] / Ignore ] ; ! not properly assimilation; \r necessary to prevent sr -> chr
define Lexicon LexiconAssimilation;

```

FIGURE 3.69: Code implementing consonant assimilation

A good argument can be made for ordering assimilation above palatalization, and some readers may wonder why I have not done so. I put palatalization first mainly for historic reasons. Part of the palatalization rule is sensitive to the boundaries of a syntactic word, whereas the assimilation rule is not. Before I discovered the ignore operator, being able to remove irrelevant characters from the lower language of the lexicon meant that my rules could be considerably simpler. So, I ordered palatalization first, followed by the removal of morpheme boundaries from the lower language, followed by assimilation. Although the historical reasons for this ordering no longer apply, the ordering persists. In (partial) consequence of this ordering, the assimilation rule has to list separate palatalized and alveolar reflexes of the same underlying phoneme.

3.8.2.27 Changing /i/ to [a] at the Beginning of Vowel Clusters

As discussed pages 11-12, /i/ at the beginning of a vowel cluster is realized as [a]. The handful of exceptions to this rule are handled in the transducer by representing the invariant /i/ as ⟨é⟩ (see discussion on transducer orthography, page 110, and rule to prevent /i/ from becoming [a] on pages 138-139). The implementation of the rule, shown in Figure 3.70 on the next page, simply replaces any ⟨e⟩ with ⟨a⟩ when the following symbol is a vowel.

```

define LexiconLowerWeakI [
  Lexicon .o.
  [ e -> a || _ Vowel ] ];
define Lexicon LexiconLowerWeakI;

```

FIGURE 3.70: Code to change /i/ to [a] at the beginning of vowel clusters

3.8.2.28 Standardizing Lower-Language Orthography

At this point, all that remains is to clean up the upper and lower languages of the transducer and to save the resulting files. This rule simply changes the transducer's lower language so that any instances of ⟨e⟩ and ⟨é⟩ become ⟨i⟩ and any instances of ⟨q⟩ become ⟨q̄⟩. The code is given in Figure 3.71.

```

define LexiconLowerStandardOrthography [
  Lexicon .o.
  [ [ e | é ] -> i ] .o.
  [ q -> q̄ ];
define Lexicon LexiconLowerStandardOrthography;

```

FIGURE 3.71: Code to convert lower-language to standard Iñupiaq orthography

3.8.2.29 Converting Upper-Language Forms to a Standard Format

Figure 3.72 shows the definitions of two rules to convert the transducer's upper language into a standard format. The first rule, `StandardTopSide`, will be used in creating a traditional lexical transducer, while the second rule `GuessTopSide`, will be used in creating a stem-guessing transducer (see Section 2.3.7 on pages 59–60 and Section 3.8.2.5 below).

```

define StandardTopSide [
  [ "+Pro" <- "+N" || .#. s u MorphemeBoundary \"+V"* _ ] .o. ! treat "su" as a pronoun, not a noun
  [ i <- e ] .o.
  [ i <- é ] .o.
  [ q <- q̄ ] .o.
  [ 0 <- DeleteOnTop ] .o.
  [ 0 <- GEM ] .o.
  [ 0 <- OPTGEM ] .o.
  [ 0 <- Consonant || _ Vowel (Consonant) HISTORICCONSONANT ] ];

define GuessTopSide [
  [ 0 <- DeleteOnTop ] .o.
  [ 0 <- Consonant || _ Vowel (Consonant) HISTORICCONSONANT ] ];

```

FIGURE 3.72: Code to standardize the transducer's upper language

All of the code in `GuessTopSide` is also found in `StandardTopSide`: the deletion of “historic” consonants (see example (23) and discussion on pages 17–18)²³ and a number of non-alphabetic symbols (`DeleteOnTop`). `StandardTopSide` makes a number of additional changes, converting the symbols for /i/ and /q̣/ to standard orthographic forms ⟨i⟩ and ⟨q⟩, respectively; deleting rule triggers for mandatory and optional gemination; and changing the tag +N to +Pro following the stem *su-* ‘what’ (*su-* is listed in the lexicon as a noun stem because it can take nominal suffixes, but this rule changes the grammatical tag that appears with it to +Pro because of its pronominal semantics). These additional changes bring upper-language strings in line with standard Iñupiaq orthography and other conventions. However, for an analysis involving a guessed stem, it may be helpful to know that the guess contained a “strong” ⟨q⟩ or that a suffix in the hypothesized word can trigger gemination, so these details are preserved on the analysis side.

Unlike most of the rules, the definitions of the top-side rules do not contain composition with the rest of the transducer; this is because there are two of them and only one will apply to each version of the transducer (guessing or non-guessing).

3.8.2.30 Saving the Compiled Transducers

The final step is to compose the current state of the transducer with each of the upper-language cleanup rules described in Section 3.8.2.29, together with filters to either disallow guessing or to disallow predefined stems, and to save the resulting compiled transducers in XFST binary format. Figure 3.73 contains the code to accomplish this.

```
read regex [ GuessTopSide .o. $" +Guess" .o. Lexicon ];
save stack ipk_guesser.fst
pop

define Top [ StandardTopSide .o. ~$" +Guess" ];
read regex [ Top .o. Lexicon ];
save stack ipk.fst
```

FIGURE 3.73: Code to finalize the upper languages and save the guessing and non-guessing transducers

The guessing transducer is defined first; the filter `$" +Guess"` ensures that only stems generated by the guessing rules will be included in this transducer. The command `read regex` pushes the new transducer onto the XFST stack as soon as compilation is complete, and then the command `save stack` saves it under the filename `ipk_guesser.fst`. The command `pop` clears this transducer off the stack so that it is not re-saved during the subsequent operation.

Before finishing and saving the non-guessing transducer, a convenience variable, `Top`, is defined; this is to facilitate debugging, as any of the previously defined crumb variables can be composed below this variable to create a transducer with a standard upper language and an intermediate lower language. The filter `~$" +Guess"` removes all “guessed” stems from the transducer. After this definition, the non-guessing

²³In its current implementation, the guesser does not generate stems with historic consonants, but this functionality could be added in the future.

transducer is compiled and saved under the filename `ipk.fst`. The non-guessing transducer remains on the stack so that developers can interact with it if they so desire. To interact with the guessing transducer, one need only enter `load ipk_guesser.fst` at the XFST command prompt.

Chapter 4 Evaluation

The transducer was evaluated for coverage and accuracy in recognizing Iñupiaq words. Evaluation data came from a small corpus of texts in the North Slope dialect; this corpus is described in Section 4.1. Evaluation procedures are described in Section 4.2, and results are given in Section 4.3.

4.1 Corpus of North Slope Dialect Texts

In order to facilitate the development of the Iñupiaq transducer as well as future projects, Lori Levin of Carnegie Mellon University arranged for students Ida Mayer, J. Eliot DeGolia, Sai Venkateswaran, Paul Lundblad, and Shinjae Yoo to type a set of texts written in the North Slope dialect and published by or in conjunction with the Alaska Native Language Center. The following texts were included:

- *Aḡuḡhuyuk* by Henry Nashaknik (1973)
- *Ataatalugiik = Grandchild with Grandfather* by Vincent Nageak (1975)
- *Avaqqanam Quliaqtuaqtanjik: I. Qupquḡiaq II. Iñuqutḡigaurat = I. The Ten-Legged Polar Bear II. Dwarves* by Floyd Ahvakana (1975)
- *Iqiasuaq Aviḡḡaq = The Lazy Mouse* by Elsie Mather (1973)
- *Jennie-m Iñuugignija Fairbanks-mi = Jennie’s Life in Fairbanks* by Ingrid Herreid (n.d.)
- *Malḡuk Quliaqtuak: Aahaalliglu Piyaanḡiḡlu / Aniqpaktuuaq Aviḡḡaq = Two Stories: The Old Squaw and Its Ducklings / The Large Lemming* by Harold Kaveolook (1974)
- *North Slope Iñupiaq Dialogues: Supplement to North Slope Iñupiaq Grammar: First Year* by Edna Ahgeak MacLean (1985)
- *Quliaqtuat Mumiaksrat: Iḡisaqtuanun Savaaksriat* by Edna Ahgeak Maclean (1986c)
- *Savaktugut sulī Piuraaqtugut = We Work and We Play* by Marie N. Blanchett and Martha Teeluk (1973)
- *Taiḡuallarḡa = I Can Read* by Martha Teeluk and Marie Blanchett (1973)
- *Tikiḡaḡmigguuq.... = In Point Hope, Alaska....* by Floyd Ahvakana (1973)

An automatic count indicates that the corpus contains 9647 tokens representing 5662 unique types.¹ The contents of Kaveolook (1974) (about 250 tokens) were used for development; the remaining texts were split automatically into sentences by Shinjae Yoo, and the sentences were randomly divided into development and test datasets of approximately equal sizes. The test dataset was set aside for the development of a “gold standard” set of morphological analyses, and I was not allowed to see it.² From the development dataset I reserved 500 randomly-selected tokens, representing 448 unique types, to be used for evaluation in this thesis. The remainder of the development dataset (4406 tokens, 2887 types) constituted the set of Iñupiaq words I was allowed to see and use to improve the performance of the transducer throughout development.

¹Tokens were counted with Tcl 8.6b1.2 using the command `regexp -all {[:alpha:]-} [string tolower $corpuscontents]` (where `$corpuscontents` contains the complete text of the corpus). This pattern recognizes all consecutive, maximally long strings of alphabetic characters and/or hyphens. Types were also counted with Tcl, using the command `llength [lsort -unique [regexp -inline -all {[:alpha:]-} [string tolower $corpuscontents]]]`.

²Because the source documents contained testing data, I was also not allowed to view those documents.

4.2 Evaluation Procedures

As mentioned above, 500 tokens were set aside for evaluating the transducer. A total of 433 unique types were present in this data. These were sent through the transducer using the XFST lookup utility (see Beesley 2004b) invoked with the lookup-strategy script given in Figure 4.1. This script instructs lookup to attempt lookup using the non-guessing transducer first, and if this fails, to try the stem-guessing transducer. The transducer which recognized a given word could propose one or several parses, but the lookup-strategy script that was used precluded the possibility that both the guessing and non-guessing transducers propose parses for the same type; in other words, if a word was parsed by the non-guessing transducer, no guesses were proposed for that word, and vice-versa. The presence or absence of the tag +Guess in a parse indicated whether the word was handled by the non-guessing transducer or the stem-guessing transducer (see page 106). A handful of types were not recognized by either transducer; these were clearly identified in the output of the lookup utility.

```
nonguessing ipk.fst
guessing ipk_guesser.fst

nonguessing
guessing
```

FIGURE 4.1: Lookup-strategy script used for evaluating the transducer

The 500 tokens were divided into three categories, depending on whether they were recognized by the non-guessing transducer, the stem-guessing transducer, or neither; the number of tokens in each category was taken as a measurement of recognition. Accuracy was assessed using random samples of each of these categories: 100 tokens recognized by the non-guessing transducer, 50 “guessed” tokens, and all tokens rejected by both transducers. Evaluation procedures were slightly different for each category:

- Recognized tokens: Lawrence Kaplan rated each parse as correct or spurious. In cases where several possibly correct parses were proposed for a single token, he also distinguished likely parses from less likely ones. For tokens where all parses were incorrect, he provided a correct morphological analysis.
- 50 guessed tokens: Lawrence Kaplan, in consultation with UAF Iñupiaq instructor and native speaker Ronald Brower, provided a morphological analysis of each token; these analyses were checked against the transducer’s guesses. The reason for using a different approach for guessed tokens than for recognized ones was the sheer number of guesses proposed by the transducer. In preliminary tests using the development dataset, the non-guessing transducer generated an average of 2.9 parses per recognized type; in contrast, the stem-guessing transducer proposed an average of 49.7 parses per guessed word. It seemed less burdensome to ask the human rater to generate a few correct parses for these types than to wade through dozens of incorrect ones.
- Unrecognized tokens: I examined each to determine why it was not recognized. For words which appeared to be legitimate Iñupiaq words, Lawrence Kaplan provided a morphological analysis, in

consultation with Ronald Brower.

4.3 Results

4.3.1 Recognition

Recognition results, not taking accuracy into account, are given in Table 4.1. The majority of test words were recognized by the non-guessing transducer, and most of the remaining words were tackled by the guesser. It's interesting to note that the type-to-token ratios for unrecognized and guessed words are 1:1, whereas the ratio for the recognized words is closer to 5:6. In other words, all types occurring twice or more in the test data were recognized by the non-guessing transducer. This fact tentatively suggests that the more common a word is, the greater the likelihood that the transducer will recognize it (two caveats: first, unfortunately, the ability to recognize common words is worth less for a language like Iñupiaq, where very few words can be considered "common", than in a morphologically poor language like English; second, the small size of the corpus and the small number of texts used to create it probably skew the type-token ratio, artificially inflating the frequency of words central to the theme of each text).

TABLE 4.1: Recognition results, not taking accuracy into account

Category	# Tokens	% Tokens	# Types	% Types
Recognized words	406	81.2%	339	78.3%
Guessed words	84	16.8%	84	19.4%
Unrecognized words	10	2.0%	10	2.3%

Table 4.2 on the next page gives statistics related to the number of unique parses generated by the transducer. The first row gives statistics relating to the non-guessing transducer, for the types it was able to parse. The second row gives statistics for the parses generated by the stem-guessing transducer for types which it was able to parse and which the non-guessing transducer was unable to parse. To provide a better understanding of the guesser's performance, two additional rows are given. The first contains the results of applying the stem-guessing transducer to the set of words that were successfully recognized by the non-guessing transducer (the guesser was unable to recognize five words from that set). The second contains the guesser's results when given as input the union of the first two sets of words (minus the five words it could not recognize).

It's unsurprising that the non-guessing transducer would generate fewer parses per type than the guessing transducer; the former is forced to choose from among the stems it knows, while the latter is free to imagine that any phonotactically correct string may be the stem, as long as it can provide an accounting (likely or otherwise) for the rest of the string. What is remarkable is the magnitude of the difference between the number of parses proposed by each transducer. It's also interesting that the guesser proposed fewer parses on average for words which the non-guessing transducer succeeded in recognizing. This difference may be explained in part by differences in the average length of types in the recognized set (9.7 characters) vs. the guessed set (14.2 characters), as there is a modest correlation between type length

TABLE 4.2: Parses per word, by category

Category	Total # parses	Min. parses per type	Max. parses per type	Avg. parses per type	St. dev. parses per type
Recognized words	912	1	35	2.69	3.17
Guessed words	4,194	2	450	49.93	81.67
Recognized words as if guessed (5 types unrecognized)	9,925	1	362	29.72	42.92
All guessable words as if guessed	14,119	1	450	33.78	53.52

and the corresponding number of guesses proposed by the transducer (Pearson's correlation = 0.473). This relationship is represented in Figure 4.2. It's apparent from this figure that the numbers in Table 4.2 are skewed upward by a handful of outliers which, for whatever reason, cause the guesser to generate hundreds of parses.

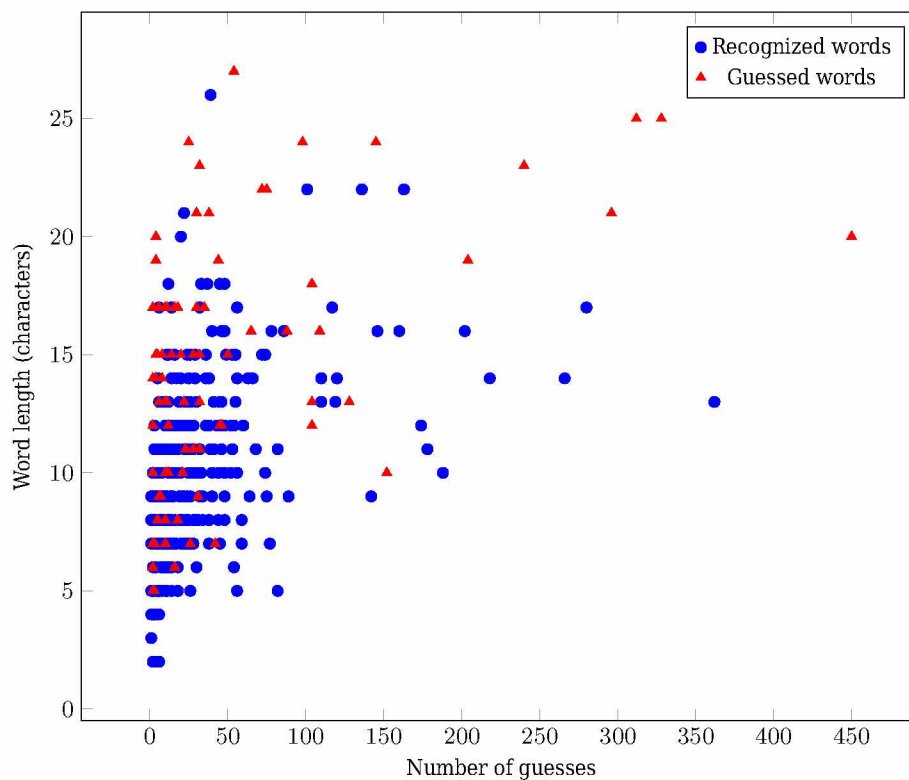


FIGURE 4.2: Guesser results: number of guesses plotted against word length in characters

4.3.2 Accuracy of Parses Not Involving Guessing

4.3.2.1 Characteristics of the Sample to Be Analyzed

The 100 recognized tokens in the sample represented 98 distinct types. Two types corresponded to two tokens each: *uvva* ‘here’ and *aṅayuqtiq* ‘first born; someone’s elder’. The first is certainly a common word in Iñupiaq, but the second must be orders of magnitude less common, and its double occurrence in a sample of 100 words highlights the fact that a corpus as small as the one from which these words were taken must be somewhat unbalanced.

Table 4.3 contains parse statistics for all the recognized types in the test data as well as for the subset that will be used to assess the transducer’s accuracy. A comparison of the two rows serves as a rough measurement of the representativeness of the sample. Figure 4.3 gives a visual representation of the data behind these statistics. The mean number of parses per type for the analysis subset is close to that for all recognized words, though the subset’s standard deviation is a bit lower. As can be seen in the figure, the curve of the subset follows a trajectory similar to that of the larger set, with minor but notable exceptions at three and ten parses per type; also, only one outlier (out of eight) in the 11–35 parses-per-type range was sampled.

TABLE 4.3: Comparison of parse statistics: all recognized words vs. subset to be analyzed for accuracy

Category	Total # parses	Min. parses per type	Max. parses per type	Avg. parses per type	St. dev. parses per type
All recognized words	912	1	35	2.69	3.17
Subset to be analyzed	238	1	16	2.43	2.59

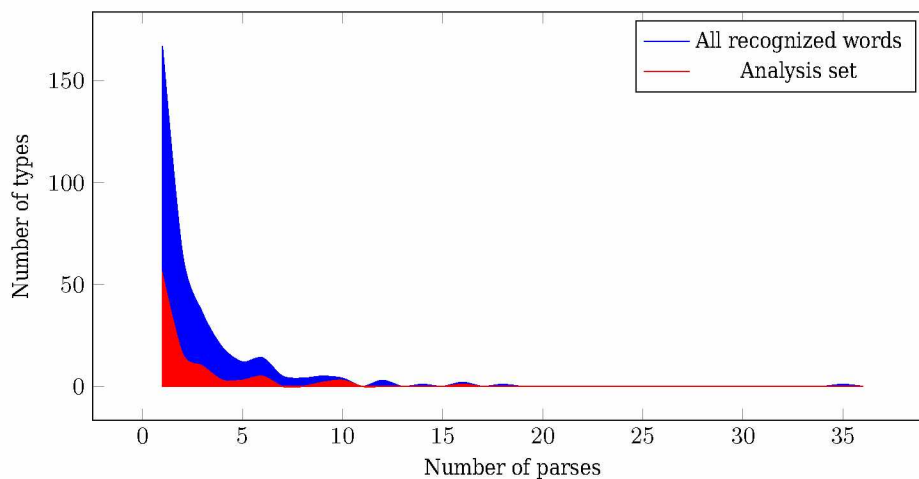


FIGURE 4.3: Parses per recognized type: all recognized words vs. subset to be analyzed for accuracy

4.3.2.2 Results

Only in the case of one of the 98 analyzed types did the transducer fail to propose a single correct parse. The failed type was *taimmauganigli* ‘let it be like that’; the cause of this failure will be examined in Section 4.3.3.

Of the 238 parses, 76 (31.93%) were judged incorrect, while another six (2.52%) were considered highly unlikely. Of the incorrect parses, 44 (57.89%) were caused by a small set of entries in the noun inflection file which were incorrectly listed as possessive. Another 14 incorrect parses (18.42%) occurred because the transducer incorrectly identified a ⟨u⟩ as the postbase *-u-* ‘to be; to be ____ing’; for example, *iñuqaǵuurut* was incorrectly parsed as shown in example (71.a); the correct analysis, which the transducer also produced, is given in (71.b).

- Ex. (71) a. **iñuqaq-suk-u-rut*
 *have.visitors-want.to-be.Ving-IND.3PL
 *‘they are wanting to have visitors’
- b. *iñuqaq-suu-rut*
 have.visitors-already-IND.3PL
 ‘they already have visitors’

The 18 remaining incorrect parses (23.68%) consist of strings of morphemes which are semantically implausible. For example, *nalugaak* was incorrectly parsed as in example (72.a), as well as being correctly parsed as in (72.b); *kiisaimma* triggered two incorrect parses, as seen in examples (73.a) and (73.b), as well as the correct parse given in example (73.c).

- Ex. (72) a. **naluk-taǵaaq-k*
 *toss-thing.used.for-abs.du
 *‘a couple of tossing things’
- b. *nalu-kaak*
 be.ignorant.of/be.uninformed.about-IND.PRS.3DU>3SGO
 ‘the two of them are ignorant of/uninformed about him/her/it’
- Ex. (73) a. **kii=taimma*
 *see.there=then
 *‘see there! then’
- b. **kii-ta=imma*
 *clench.teeth-OPT.1PL=the.aforementioned.one
 *‘let’s all clench our teeth the aforementioned one’
- c. *kiisaimma*
 finally
 ‘finally!’

The parses incorrectly identified as possessive should be easy enough to eliminate, since they result

from mistakes in the lexicon. The other incorrect parses are harder to deal with, since their incorrectness comes from semantic considerations, which are beyond the scope of the transducer. Some common faulty parses might be filtered out of the lexicon; for example, the dubious morpheme sequence **-suk-u-*, which appears in (71.a) on the preceding page and eight other incorrect parses, could be prohibited. But such filtering must be done with care so as to prevent legitimate morpheme combinations from being accidentally removed from the lexicon.

At present, it seems necessary to accept the possibility that a few nonsensical analyses may be proposed for any given word. Human users of the transducer will need to be aware of this fact, and any software which relied on the transducer for morphological analysis would need a way to rank parses in terms of likelihood. Although one would hope that both humans and software using the transducer would take context into account when evaluating parses, there is a simple, context-free heuristic which might point them in the right direction: gravitate toward the parses with the fewest morpheme boundaries. While not infallible, this heuristic is generally correct, in part because it favors analyses with lexicalized compounds over analyses containing the constituent parts of these compounds (e.g., *iñupiaq* rather than *iñuk-piaq*) and in part because it disprefers analyses containing short formatives like the postbase *-u-* which, compared to longer formatives, are more likely to appear in incorrect analyses.

I measured the accuracy of this heuristic using Dr. Kaplan's judgments of the 238 recognized parses. Recall that Dr. Kaplan rated each parse as incorrect, unlikely, possible, or likely. In the case of nine types (twelve tokens) where Dr. Kaplan ranked two parses as possible, and the parses were identical except that one contained a lexicalized compound where the other contained the parts of this compound as separate morphemes, the parse with the lexicalized compound was given precedence over its counterpart. (Admittedly, this modification to the parse rankings biases the results in favor of the heuristic, but given the equivalence or near-equivalence of the pairs of parses affected by these modifications, it should not be terribly controversial.) In none of these cases did such a change in precedence cause a particular parse to outrank parses implying a substantively different analysis. The 44 parses caused by inflectional entries incorrectly listed as possessive were excluded from the analysis, since the mistake which caused them to be generated is easily rectified. All other incorrect parses were included in this analysis.

The results of this analysis are shown in Table 4.4 on the following page. The results look a bit more promising than perhaps they really are, since for 56 of the 82 types where the heuristic succeeded, the transducer provided only one parse to choose from. The fact that the heuristic failed partly or completely in 16.33% of all evaluated cases should caution transducer users against relying solely on it to identify correct parses; still, the results suggest that considering the number of morpheme boundaries can be an effective strategy in evaluating transducer output, when used alongside other considerations.

4.3.3 Analysis of Failures of the Non-Guessing Transducer

This section will examine 61 types and the factors that prevented the non-guessing transducer from proposing correct analyses for them. The types are drawn from the data sets selected for analysis, as follows: the one type from the recognized words for which no correct analysis was proposed; all 50 guessed types; and all 10 types rejected by both the guessing and non-guessing transducers. For each of these types, Lawrence Kaplan, in consultation with UAF Iñupiaq instructor and native speaker Ronald

TABLE 4.4: Result of automatically selecting parses with fewest morpheme boundaries

Outcome	Types
All likely parses selected; no other parses selected	82
All likely parses plus one or more less likely or incorrect parses	9
Some (not all) likely parses selected; no less likely or incorrect parses selected	4
Only one of two likely parses selected; one incorrect parse selected	1
Less likely parse selected; likely parse not selected	1
No correct parses; incorrect parse selected	1

Brower, attempted to propose a correct morphological analysis. I checked the transducer’s lexicon and rules against each of these analyses and found a total of 73 causes of failure. Most types failed because of a single factor; in thirteen cases, there were two factors preventing a type from being recognized. The causes of failure are given in Table 4.5; these are discussed in greater detail in the following subsections. Bear in mind that because of the sampling methods used to gather the words analyzed here, the numbers in this section are not necessarily proportional to either the test data set or the set of words not recognized by the non-guessing transducer; the most that can be said is that strong trends in the data below are likely to hold for the entire set of test data (for example, there will be more failures due to postbases missing from the lexicon than due to stems missing from the lexicon).

TABLE 4.5: Causes of non-guessing transducer failure

Cause of failure	Total	Misrecognized	Guessed	Unrecognized
Postbase not in lexicon	19		19	
Typo	13		10	3
Phonological rule inadequate	5		5	
Stem not in lexicon	5		4	1
Unusual inflectional morphophonology	5		5	
English stem	4			4
Phonological rule missing	3		3	
Postbase inadequately specified	3		3	
Stem variant not in lexicon	3		3	
Inflectional ending inadequately specified	2		2	
Morphotactics inadequate	2	1	1	
Postbase allomorph not in lexicon	2		1	1
Reduced form not in lexicon	2		1	1
Stem incorrectly specified	2		2	
Enclitic not in lexicon	1		1	
Flaw in postbase file long distance dependency logic	1		1	
Stem not specified as transitive	1		1	

4.3.3.1 Stem-Related Failures

Fifteen failures (20.55%) had to do with stems. Four stems were English (*Christina*, *claim*, *mining*, and *tape*). Five Iñupiaq stems were missing from the lexicon; among these, one was the cheer *yai-ai!*, and the

other three were names of people. These sorts of failures were expected. The fifth missing stem, *sivunnigi-* ‘be determined’, was not the sort of stem that was expected to be missing, although it is also not listed in MacLean (n.d.a).

Three other stem failure categories were also not expected. Three unrecognized stems were variants of stems which are found in the lexicon: *naiǰli-* is a variant form of *naikǰi-* ‘become shorter’, and *tasama* and *tauna* are variants of demonstratives *tasamma* ‘down there’ and *taunna* ‘that down there’. One stem was not recognized because the lexicon does not list it as a transitive verb; one stem was incorrectly listed as ending in /ǰ/ when it actually ends in /ǰ/; and one stem, that of duale tantum noun *kamiktuuk* ‘pants’, was specified with a singular stem which, while theoretically possible, is rarely if ever used in reality.

4.3.3.2 Postbase-Related Failures

Issues related to postbases constituted 34.25% of all failures (25 in all). In fact, the number one cause of failure was postbases missing from the lexicon. Of the 16 absent postbases, nine are listed in MacLean (n.d.a) as “limited” (which I had interpreted to mean unproductive); the other seven are not listed in the dictionary. Two postbases occur more than once in the guessed words: *-si-* (which can mean ‘become’ or which can act as a detransitivizer) occurs three times, and *-uti-* (which may indicate that the action denoted by the stem is done along with or for the benefit of another, or that multiple agents perform the action for or to each other) occurs twice.

Other problems with postbases included:

- *postbase inadequately specified*: two instances of the postbase *-aqsi-* ‘be about to, be going to ____; continue to ____’ were preceded by /i/ realized as [i], suggesting that this postbase is (or can be) an exception to the rule /i/ -> [a] / __ V (see pages 11–12). A type containing postbase *-lgisaq-* ‘which is being or has been ____ed again’ was not recognized in part because this postbase appears to require special endings.
- *postbase allomorph not in lexicon*: postbase *-suga-* ‘don’t you remember that ____; to unexpectedly, surprisingly ____’ appeared as *-uga-* following a /q/; this suggests that (at least for some speakers) this postbase follows the same allomorphy pattern as *-suk-* ‘want to’, namely that the initial consonant is dropped when the preceding segment is /k/ or /q/. Postbase *-lit-* ‘to provide with a ____’ appeared as *-it-* with the equals pattern (see pages 135–136) and gemination of the preceding consonant. This allomorph was not in the lexicon or listed in the dictionary.
- *flaw in postbase file long distance dependency logic*: a rule in the LongDistanceDependencies section of the postbase file contained a bug. The purpose of the rule is to clear valence restrictions for deverbal postbases. The rule identifies these as belonging to a certain class of category codes and having a continuation class not in a certain set of classes. The set of classes did not include a handful of classes designed to restrict the allowable moods that could co-occur with certain postbases. Postbase *-taa-q/-laaq-* ‘going so far as to ____’ requires conditional, consequential, or contemporative I mood endings, and because these classes weren’t listed in the rule, the postbase was filtered out of the lexicon.

4.3.3.3 Inflection-Related Failures

Seven failures (9.59%) had to do with inflection. Five of these are from morphophonological results not predicted by the transducer's rules (or by the rules described in MacLean's grammar books), but which are not necessarily incorrect. One such word is *nukatchia* 'his/her younger sibling', where the root is *nukatchiaq* and one would have expected *nukatchiaŋa* in the ABS.SG.3SGP form. Another is *tuvaqanni* 'his/her companion, partner', where the expected form would be *tuvaqatini*. Both of these are older forms, and while it's possible to guess at the rules responsible for them, more research would be required to determine the limits of those rules before they could be added to the transducer with any degree of confidence.

Two other inflection-related failures stem from inflectional endings specified in such a way that some variants are not recognized. For example, the word *aŋayuqaŋiŋñun* 'to his/her parents' was rejected because the TRM.DU.3SGP ending was specified as *-ŋiŋñun* rather than *-ŋiŋñun*. Because the transducer's assimilation rules will optionally change ⟨g⟩ to ⟨ŋ⟩ before a nasal, specifying the ending with a ⟨g⟩ would have allowed either variant of the ending to be recognized.

4.3.3.4 Enclitic-Related Failures

Three failures come from enclitics missing from the lexicon; two of these (*suli* 'and furthermore; also; still in progress' and *ki* 'come on; go ahead') are reduced forms, and a third (*ami*) appears to be just an enclitic.

4.3.3.5 Failures Due to Phonological and Morphotactic Rules

Four failures stem from the rule which changes underlying /i/ to [a] when another vowel follows. In each case, /i/ occurs to the left of an enclitic (or reduced-form) boundary. For example, the word *qaŋaliukuak* 'since long ago', which contains enclitic =*li* and reduced demonstrative pronoun *ukuak*, was not recognized because the transducer expected **qaŋalaukuak*. The rule needs to be modified so that it only operates within syntactic words.

Another failure resulted from an oversight in the palatalization and assimilation rules, which erroneously change /t/ to [n] (rather than [ñ]) in the environment /i/ — N (where N represents a nasal consonant). If the palatalization rules were ordered lower in the cascade than the assimilation rules, this error would not have occurred.

Three other failures were due in part to the absence of rules. The word *kaviuŋiaqsigaigña* 'he/she became envious of the other ones' consists of *kaviuŋiaqsigaa* and cliticized demonstrative pronoun *igña*; the transducer currently lacks a rule that would delete the second vowel of a word-final vowel cluster when the word is followed by a vowel-initial enclitic or reduced form. Words *natiŋiniqqami* and *tuvvamun* are conservative forms which illustrate a rule not currently implemented in the transducer where oblique case endings optionally trigger gemination.

Two failures are attributable to overly restrictive morphotactic constraints. In both cases, the stem is a demonstrative and is followed by a postbase which is currently listed in the lexicon as attaching (only) to noun stems.

4.3.3.6 Failures Due to Typographic Errors

Thirteen failures (17.81%) were due at least in part to typos in the data; some of these were introduced in the typing of the corpus, but several are present in the source texts.

4.3.3.7 Dealing with Causes of Failure

Some of the failures reflect flaws in the transducer which are easily fixed. Others are due to bad input and need not be addressed. Still others reflect a gray area where attempting to repair the failure could lead to unintended consequences; for example, adding unproductive or marginally productive postbases to the lexicon or adding rules to accommodate an archaic inflection pattern may increase the number of incorrect parses generated by the transducer. Following is a category-by-category analysis of what I believe should be done to address failures, and how many failures are addressable.

4.3.3.7.1 POSTBASES NOT IN LEXICON The normal method for adding postbases to the lexicon allows them to combine freely with a number of stems. This may not be appropriate for the postbases identified in this error analysis, if some or all of them are indeed limited in productivity. On the other hand, adding postbases in a way that significantly restricts their ability to combine with stems increases the complexity (and frequently, the memory requirements) of the lexicon. One reasonably cautious approach to dealing with these missing postbases involves creating three transducers: the first would contain only postbases believed to be productive, the second would contain all known postbases, and the third would have the ability to guess postbases. The transducers would be given cascading priority in an XFST lookup script like the one shown in Figure 4.1 on page 147. This would prevent less-productive postbases from adding noise to the parses of more straightforward words, but still make those postbases available as part of the transducer's second line of attack should the more restricted lexicon prove insufficient. This approach is described in more detail on page 160.

4.3.3.7.2 TYPOS Since these reflect issues external to the transducer, nothing need be done (or can be done) about them.

4.3.3.7.3 INADEQUATE PHONOLOGICAL RULES There are only two rules concerned, and there is no reason not to fix them. This would resolve five failures, although these fixes alone would only enable the transducer to recognize two of those five types (among the other three types, one contained a typo, one contained the postbase *-si-*, and one had a stem-postbase combination not allowed by current morphotactic constraints).

4.3.3.7.4 STEMS NOT IN LEXICON Three of these are names and another is a cheer; while adding these to the lexicon would probably not do great harm, it's questionable whether they really belong there. The case for adding *sivunniġi-* 'be determined' is more clear-cut.

4.3.3.7.5 UNUSUAL INFLECTIONAL MORPHOPHONOLOGY Because these words are unusual, it's not clear what should be done to deal with them. More research needs to be done to determine the extent to which the patterns exhibited by these words can be generalized to the rest of the lexicon.

4.3.3.7.6 ENGLISH STEMS None of the stems in this category are likely to be frequent enough to merit inclusion in the transducer; however, at some point in the future it might be worth developing a transducer capable of guessing foreign stems, mostly for the purpose of identifying inflectional endings and determining grammatical categories, which could facilitate syntactic analysis and lemmatization of surrounding words.

4.3.3.7.7 MISSING PHONOLOGICAL RULES These should not be difficult to fix, but caution should be taken to ensure that implementing this rule doesn't cause anything else to break. Each additional phonological rule increases the size of the transducer and the time and resources required to compile it, but the limited scope of these rules should constrain these increases to modest levels.

4.3.3.7.8 INADEQUATELY SPECIFIED POSTBASES It should not be difficult to determine if postbase *-aqsi-* always preserves the underlying quality of preceding /i/, and, if that is the case, to change the lexicon entry accordingly. Identifying the set of endings that can follow postbase *-lgisaq-* will be a more difficult task.

4.3.3.7.9 STEM VARIANTS NOT IN LEXICON As long as these variants are accepted by the Iñupiaq-speaking community, it should be no trouble to add them to the lexicon, thus resolving three failures.

4.3.3.7.10 INADEQUATELY SPECIFIED INFLECTIONAL ENDINGS These are another simple fix.

4.3.3.7.11 INADEQUATE MORPHOTACTICS Resolving these failures should not be particularly difficult, although the combinatorial limits of demonstratives and the postbases in question should be determined first.

4.3.3.7.12 POSTBASE ALLOMORPHS NOT IN LEXICON One of these (*-[s]uga-*) belongs in the main non-guessing transducer and is easy to fix; the other (*-it-*, from *-lit-*) is an allomorph of a postbase marked as 'limited' in the dictionary, and could be added to the second-priority transducer with the other less-productive postbases.

4.3.3.7.13 REDUCED FORMS NOT IN LEXICON Adding reduced forms of *suli* and *ki* should be straightforward.

4.3.3.7.14 INCORRECTLY SPECIFIED STEMS One of these (*tuvaq* 'landlocked ice') can be fixed by changing /q̄/ to /q̃/. Fixing the other (*kamiktuuk* 'pants') will require a fundamental change to the treatment of dualia tantum and pluralia tantum, since this error demonstrated an error in my understanding of these phenomena. (A less likely possibility is that the word was incorrectly transcribed; however, *kamiktuunik* is entirely plausible. I assume the error is on my part and not on the part of the transcriber.)

4.3.3.7.15 ENCLITIC NOT IN LEXICON If this enclitic (=ami) is productive, then it should be added to the lexicon, which should not be difficult. Adding this enclitic may lead to more false parses, however.

4.3.3.7.16 **FLAW IN POSTBASE FILE LONG DISTANCE DEPENDENCY LOGIC** There's no reason not to fix this, which should be simple.

4.3.3.7.17 **STEM NOT SPECIFIED AS TRANSITIVE** This is easily changed, and the fact that it occurred in the corpus as a transitive is sufficient evidence to justify editing the entry.

4.3.3.7.18 **SUMMARY** In all, 25 failures (34.25%) are easily resolved. Another 21 (28.77%) are unresolvable (typos) or unlikely to be resolved (English stems, stems not in lexicon [with the exception of *sivunnigi-*]). For the 19 failures due to unrecognized postbases and the one failure due to an unrecognized allomorph of a "limited" postbase (27.40% of all failures), I have proposed a course of action which seems promising but which remains to be tested. For the remaining seven failures (9.59%), more research needs to be done to determine how or whether to modify the transducer.

With a few notable exceptions (such as the rule incorrectly changing /i/ to [a] across an enclitic boundary), most of the factors causing recognition failure applied to only one or two words in the analysis set. For example, although many of the failures were due to postbases missing from the lexicon, most of the fifteen missing postbases only accounted for one failure each. This fact illustrates the effect of Zipf's law on transducer development: a small handful of rules and lexicon entries account for a relatively large number of words, but the remaining words require a much larger set of rules and lexicon entries, each of which applies to only a few less-frequent words. Viewed in a slightly different light, development of a lexical transducer is subject to the law of diminishing returns: as the number of rules and lexicon entries increases, the additional gains from adding another rule or lexicon entry tend to become progressively fewer.

4.3.4 Accuracy of Parses Involving Guessing

4.3.4.1 Characteristics of the sample to be analyzed

There were no duplicate tokens among the guessed words in the test data, so the 50 tokens selected for analysis represent 50 distinct types. These tokens account for nearly 60% of all recognized types, and so should give a very accurate picture of the guessing transducer's ability to parse words missed by the non-guessing transducer.

Table 4.6 on the following page contains parse statistics for all guessed types in the test data set and for the subset to be analyzed in this section. The numbers are quite similar, and although the analysis sample does not include the outlier with the most parses, it does include slightly more parses per type on average than the entire set of guessed types. Figure 4.4 on the next page displays the distribution of parses per type for all guessed words and for the subset to be analyzed. The sample appears to be a reasonable approximation of the set from which it was taken. The graph has a strikingly jagged quality to it, due in part to the small number of tokens in the sample and in part to the large number of average parses per type. Because no single number of parses per type accounts for more than seven types, any difference from one data point to the next appears as a steep slope.

TABLE 4.6: Comparison of parse statistics: all guessed words vs. subset to be analyzed for accuracy

Category	Total # parses	Min. parses per type	Max. parses per type	Avg. parses per type	St. dev. parses per type
All guessed words	4,194	2	450	49.93	81.67
Subset to be analyzed	2,747	2	328	54.94	82.91

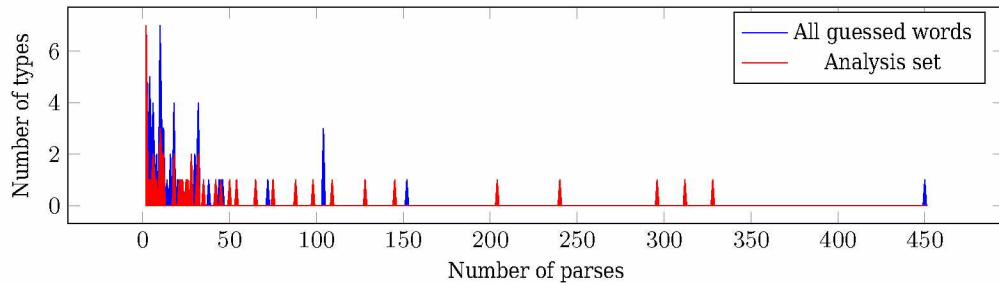


FIGURE 4.4: Parses per guessed type: all guessed words vs. subset to be analyzed for accuracy

4.3.4.2 Results

Given the information in Section 4.3.3, it should come as no surprise that the guesser produced rather dismal results. Its job, after all, is to predict stems that are missing from the lexicon, and while it succeeds for the most part in performing that task, it turns out that most of the non-guessing transducer's failures have nothing to do with missing stems. Only seven of the fifty words in the analysis sample weren't recognized by the non-guessing transducer due to stem-related issues alone. Two of these, demonstratives *tauna* and *tasama*, weren't correctly analyzed by the guesser because it is programmed to guess only noun and verb stems (in theory, these are the only open categories in the Iñupiaq lexicon). The other five (proper names *Aalagum* [stem *Aalaak*] and *Aimaġlu* [stem *Aimaq*], common noun *kamiktuuġik* [stem *kamiktuuk*] and verbs *naigġivġugich* [whose stem is a variant of *naikġi-*] and *uiññiġai* [whose stem, *uit-*, was specified in the transducer's lexicon as intransitive only]) were correctly analyzed, however. Additionally, for two words which were passed over by the non-guessing transducer because they contained *-aqsi-* preceded by */i/*, the guesser proposed analyses which were correct in every respect except that this */i/* was given as */i/* (the guesser's analyses, unlike the non-guessing transducer's, distinguish between strong and weak */i/* and */q/*). If these are counted as correct analyses, a grand total of seven types out of fifty (14%) were correctly analyzed by the guessing transducer. The stem guesser's design prevents it from correctly analyzing any word containing an unknown postbase, inflectional ending, or enclitic, nor is the transducer prepared to handle words with typos or unusual morphophonological patterns. It goes without saying that not anticipating the magnitude of these limitations represent a monumental failure on my part.

All is not quite lost, however. In some applications, an incomplete analysis is still useful. For example, in syntactic analysis, a complete morphological analysis of each word is often not needed; a word's grammatical category and inflectional information often suffice. When it comes to identifying this information,

the guesser performs somewhat better. For 32 of the 50 types (64%), it was able to supply the correct inflectional information.

Unfortunately, every correct or partially correct parse proposed by the guesser is buried in a haystack of spurious parses of varying degrees of badness. A number of factors contribute to this situation:

- Iñupiaq's word-final phonotactics are fairly restrictive, and the unpossessed absolutive and relative endings are phonologically simple, so nearly every word will be analyzed as an absolutive or relative noun (possibly among other analyses).
- There is considerable syncretism among Iñupiaq inflectional endings; this means that, for example, any word ending in ⟨t⟩ will be analyzed as an absolutive dual noun and a relative dual noun (possibly among other analyses).
- In the current transducer, the multiplicative effect of the legitimate syncretism described above on the number of parses is exacerbated by the inflectional ending entries incorrectly specified as possessive (see page 151).
- The guesser distinguishes strong and weak /i/ and /q/; for each ⟨i⟩ or ⟨q⟩ in the input, at least two parses will likely be generated (although this depends on whether the phoneme appears in an inflectional ending, a postbase or enclitic recognized by the transducer, or elsewhere).
- Optional rules (most notably, palatalization) result in two parses for every segment that could be affected by the rule.
- If the transducer recognizes a postbase which can delete segments from the preceding stem, and if there is any chance that such deletion has occurred, the transducer will propose a guess for each segment that could have been deleted.
- If a word contains, say, a stem which the transducer can guess plus a postbase and an inflection which the transducer recognizes, the transducer will also propose a set of parses treating the stem and postbase as a single stem followed by the inflection, and a set of parses with the stem, postbase, and inflection as a single (nominal) stem.

These factors conspire in a combinatorial fashion to produce a massive number of parses for all but the most simple inputs, which renders the guessing transducer (at least in its current state) nearly useless. However, I believe the notion of a guessing transducer as a fallback for a purely lexical transducer still has merit, and steps could be taken to make an Iñupiaq guesser both more accurate and more useful.

The error analysis in Section 4.3.3 demonstrated the need to consider not only unknown stems but also unknown and unproductive postbases. Instead of a two-tiered strategy (recognition, then guessing), a three- or four-tiered strategy might be useful. The first resort would be a non-guessing transducer like the current one. If this failed, the next step would be to use another non-guessing transducer containing the "limited" postbases in addition to the productive ones. Next, if needed, a phonotactically-aware transducer capable of guessing both stems and postbases could be used. Finally, as a possible last resort, one could apply an indiscriminate guesser designed to accept everything and attempt to recognize inflectional information, perhaps taking into account possible errors in the input.

The transducer at the third tier in this hierarchy would resemble the current guessing transducer, with some important differences. As noted above, it should be able to guess at postbases as well as stems, though it should probably be prevented from making both kinds of guesses for any given input, or from

guessing more than one morpheme per word. The number of parses per word could be reduced by fixing the incorrect possessive endings, eliminating distinctions between strong and weak segments, and constraining palatalization-related variation in the upper language: while the guesser's lower language should include words where palatalization has been applied incompletely or not at all, the upper (analysis) language is arguably more useful the more prescriptive it is.

Even with these changes, the transducer is likely to produce more parses than most people will find helpful. Two things could be done about this. First, a heuristic similar to the one presented for parses of recognized words (though almost certainly more complex) could be developed to identify guesses which are more likely to be accurate. This might take into account the total number of morphemes, the length of guessed morphemes, and/or the comparative rank of proposed inflectional endings within a hierarchy of likelihood (for example, if one word is parsed both as a verb in the indicative present 3rd person dual and as a noun in the absolutive dual, the verbal interpretation is most likely to be correct, though of course other factors would have to be considered).

Second, software could be developed to distill a set of guesses in order to make it easier for the user to sift through the possibilities.³ The software would take a set of guesses and display a menu containing each of the unique morphemes from those guesses. When the user selected one or more of these morphemes, the set of guesses containing those morphemes would be displayed. This would provide a simple way for the user to filter out obviously bad parses and to formulate and test hypotheses involving more promising morphemes.

³I put together a program with the basic functionality described here to help me evaluate the accuracy of the guessing transducer's output. It was extremely useful for that task, but it is not ready for prime time yet; the interface is rather crude, and at present, the program only accepts input from the database where I have stored the results of the transducer evaluation. These deficiencies could be remedied with a few hours' effort.

Chapter 5 Conclusion

5.1 Contributions and Limitations

This thesis has outlined the design and development of a lexical transducer for North Slope Iñupiaq and evaluated its ability to analyze the morphology of a variety of words taken from published texts in that dialect. The implementation of the transducer incorporates a number of innovations to facilitate development of both the lexicon and the morphographemic rules, and to allow both to be specified in a more natural way. Specifically:

- A set of unique, language-specific formats was created for specifying stems, postbases, inflectional endings, enclitics, formative classes (including epsilon continuations), conditioning environments for allomorphs, and multicharacter symbols. These formats, taken as a whole, effectively modify the LEXC language’s implicit lexical model in a way that allows the properties of Iñupiaq to be specified more naturally and directly.
 - Formatives are grouped by type (stem, postbase, inflectional ending, enclitic) rather than by a mixture of phonological and morphological criteria, as is common (and often necessary) in lexicons specified in pure LEXC. This more natural grouping allows straightforward data entry from Iñupiaq dictionaries, which is further facilitated by adopting Edna MacLean’s orthographic conventions for these files.
 - Irregular, phonologically conditioned allomorphy is common in Iñupiaq, and these formats allow all allomorphs of a morpheme to be specified together, with a common upper-language representation, formative class, and continuation class or classes.
 - Inflectional endings are specified in two-dimensional paradigms rather than in one-dimensional formative classes, making this data easier to enter, read, and maintain.
 - Reduced (cliticized) forms of stems can be specified in the stem’s entry, and will inherit the stem’s grammar tags.
 - Conceptually, formatives can belong to and/or continue to more than one formative class. This is useful when dealing with stems and postbases whose inflectional possibilities are limited to endings belonging to a particular subcategory (say, possessive nominal endings).
 - Epsilon continuations are treated as properties of formative classes rather than proper members of those classes.
 - Long-distance dependency restrictions are defined on the level of the formative class rather than on the level of individual formatives, and all long distance dependencies for a given formative type (stem, postbase, or inflectional ending) are grouped together, making them easier to manage.
- The “trail of crumbs” idiom was devised for the morphographemic rule file. This idiom can be considered an extension of what might be called the one-variable-per-rule idiom (for an example of this, see the second Southern Portuguese Pronunciation example in Beesley and Karttunen [2003:471–473]). Where one-variable-per-rule defines a variable for each rule, with a rule cascade at the end of the file, the trail-of-crumbs idiom integrates the cascade into the rule definitions so that each rule is added

to the transducer as it is defined, leaving a “crumb” at each step in the compilation process which contains the state of the transducer at that point. While it’s possible that others have conceived of this idiom before, I have not seen it described or used elsewhere. Among its advantages:

- It allows XFST to provide feedback at each step in the compilation process, making it possible to gauge the time and memory required to add each rule to the transducer and to measure the impact of each rule on the size of the transducer.
- The crumbs make it easier to pinpoint the origin of errors; users can apply the same input to multiple crumbs and easily identify the point at which the output deviates from what is expected.
- The idiom’s structure couples the definition of a rule to its position in the cascade. This means, among other things, that moving a rule’s definition relative to other definitions automatically changes the order in which it is added to the transducer. At first blush, having the location of a definition automatically linked to its position in the cascade may not seem like a big deal, but it is useful both when developing rules and when consulting the file later. A rule must take into account the state of whichever of the transducer’s languages it will modify. Most rules are composed “below” an existing transducer, modifying its lower language, so listing each rule definition directly after the rule which most recently affected the lower language makes it easier to keep the current state of that language straight, and if one needs to be reminded of changes made by earlier rules, the entire evolution of the lower language can be traced simply by looking at each rule in order. In transducers where rule definitions are separated from the rule cascade, rules are often developed in this order, but if rules are later reordered in the cascade, either rule definitions must be reordered to match (in which case a single rule reordering necessitates movements in two separate parts of the code) or there must be a mismatch between the cascade and the order of the rule definitions (which can make the file harder to read, which in turn can make it more difficult to add or modify rules in the future, since determining the state of the transducer’s languages at a given point in the cascade requires that the rules be reviewed in a non-linear order).

The transducer described here actually consists of two transducers. The first, which takes precedence over the other in analyzing words, relies on its lexicon to generate analyses. The second is able to guess stems which conform to North Slope Iñupiaq phonotactic constraints. In testing, the non-guessing transducer was able to recognize 81.2% of all test tokens and 78.3% of all test types, and of the 100 recognized tokens analyzed, a correct analysis was generated in 99 cases. The guessing transducer fared considerably worse; although it was able to propose analyses for 89.4% of the words missed by the non-guessing transducer, an examination of parses for 50 of these words revealed that most of the guesses were incorrect. Only for 14% of the examined words was the guesser able to propose a completely accurate parse, although in 64% of cases it was able to propose at least one parse with the correct inflectional information. An analysis of types which were not recognized by the non-guessing transducer revealed a number of areas where the transducer can be improved. I proposed shifting from a two-transducer strategy (non-guessing and guessing) to a three- or four-transducer strategy. The first transducer would be similar to the current non-guessing transducer. The second transducer would have a lexicon containing less productive postbases. The third transducer would be able to guess not only unknown stems but also unknown postbases.

A possible fourth transducer would accept all input and give highest priority to attempting to recognize grammatical category and inflectional information, perhaps taking into account the possibility of typos or other errors in the input. I hypothesize that this approach will be able to accurately handle many of the issues that caused words not to be correctly recognized by the current transducer.

The transducer contains few mechanisms to prevent stems and postbases from combining in semantically improbable ways, and this situation is not likely to improve in the near future. Users of the transducer will need to bear this limitation in mind. The transducer cannot, and is not intended to, replace human judgment. However, the transducer can still be a useful tool, both on its own and as a component in other software.

5.2 Possible Improvements to the Implementation of the Transducer

5.2.1 Specification of Inflection

While I believe the transducer's current format for specifying inflection represents a tremendous improvement over the traditional, one-dimensional formative class listing, it was still less-than-ideal for some aspects of Iñupiaq inflection, particularly because in many cases the endings had to be rearranged from the order in which they were given in MacLean's grammar books. It would be nice to have the flexibility to indicate not only the grammar tags belonging to each axis, but also the order in which those grammar tags should combine. Another thing that would be useful would be to replace the rigid two-dimensionality of the model with a flexible dimensionality. This would improve the description of possessive noun endings and transitive verb endings, both of which are presently contorted into a pseudo-three-dimensional format.

5.2.2 A Preprocessor for xfst

While xfst is a powerful language, it is not necessarily a perfect one. In one FAQ, Lauri Karttunen characterizes the xfst interpreter as "a carefree, hedonistic interpreter that lives for the moment. It greedily interprets each regular expression as it is parsed, based on the state of the symbol table at that very moment."¹ That greedy interpretation based on the symbol table becomes a frequent source of frustration during rule development. Three issues in particular kept cropping up:

1. If a rule fails to anticipate the presence a flag diacritic or other multicharacter symbol, strings containing those symbols will not be handled correctly by the transducer. Particularly with flag diacritics, this kind of bug is very hard to pinpoint, because xfst normally renders flag diacritics invisible to the end user.
2. If an xfst variable name or flag diacritic contains a typo, xfst simply interprets it as a new multicharacter symbol. These errors are easier to identify than the flag diacritic errors just mentioned, because sooner or later the funny symbol appears in the xfst output, but it may take a while to realize that the symbol is appearing and to identify its origin.
3. Failure to put whitespace between two or more consecutive symbols in a rule causes xfst to interpret the symbols as a single multicharacter symbol. Although symbols are separated by whitespace in

¹<http://www.stanford.edu/~laurik/fsmbook/faq/forward-reference.html>

xfst rules, they are not separated in xfst output; this makes errors of this sort more insidious than the typo errors described above. It's easy to see that there's an unwanted multicharacter symbol `Vowl` in a word like `nauVowlruq`; it's impossible to see that there's an unwanted multicharacter symbol `ch` in a word like `iñuich`. The presence of such symbols can be detected by examining the transducer's alphabet, but usually, nothing in the output of a bug like this would suggest to the developer the need to examine the alphabet.

Simple awareness of these three types of issues can go a long way toward finding and resolving them, but I would prefer to have a way for them to be called to my attention as soon as xfst encounters them in my rules. Unfortunately, xfst's "hedonistic" semantics make this impossible. The first issue can be worked around using ignoring and the elimination of flag diacritics, techniques which I have used to full advantage. The remaining issues are more difficult to deal with in xfst, and likely impossible to catch automatically.

However, it should be possible to develop a preprocessor which could take an xfst file and the multicharacter symbol declaration file and identify any places within an xfst regular expression where two or more non-syntactic symbols are strung together unexpectedly (meaning not in a variable definition earlier in the xfst file and not anywhere in the multicharacter symbol file). The developer could then be immediately alerted both of an error's existence and its precise location, instead of being left to discover a problem hours or even days later and having only a vague idea regarding what code might be faulty.

5.2.3 Separate Permissive and Prescriptive Transducers

Antonsen et al. (2009:12) describe a set of lexical transducers used for computer-based language teaching activities for Sámi. Among the goals motivating their software design were maximally tolerant recognition of student input, and prescriptive generation of output. To make this possible, they generate two separate versions of their transducer, one for recognition, the other for generation. Many of the same issues that motivated this two-pronged approach for Sámi also apply to Iñupiaq. The transducer described in this thesis does a good job of being permissive, but a prescriptive version could be created by enforcing palatalization and other phenomena treated as optional and by marking particular inflectional endings as standard and removing others from the prescriptive transducer's lexicon.

5.3 Possible Applications of the Transducer

5.3.1 Computer-Assisted Language Learning

Plans are underway to develop dynamic language lessons in which the Iñupiaq transducer generates material for the language student to respond to and then parses the response. A prescriptive version of the transducer would be ideal for helping students practice applying morphophonological rules and producing appropriate inflectional endings for different situations. The lack of computerized tools for syntactic and semantic analysis and discourse management make it more challenging to use the transducer for more communicative tasks, but with some ingenuity and some sensible restrictions on the possible range of input and output (such as the use of controlled vocabularies), it should be possible to use the transducer in interesting ways that promote more natural communication.

One activity currently under development will present a dynamically generated family tree and prompt

the user to describe various relationships. Iñupiaq kinship terminology makes finer distinctions in most cases than English kinship terminology does, and this activity will provide valuable practice both with the terminology and with the grammatical constructs commonly used in conjunction with that terminology: numbers, possessive endings, the postbase *-qaq-* ‘have’, indicative and interrogative verb endings, and the modalis case (which is applied to numbers when expressing how many of something one has).

David Adamson, a graduate student at Carnegie Mellon University, is investigating the use of the transducer as the lexical component of a Scrabble-style game intended for language learners.

5.3.2 Enhanced Dictionary Interface

Poser (2002) and Maxwell and Poser (2004) make a strong case for using morphological analysis to facilitate dictionary lookup for languages with complex morphology, including Athabascan and Semitic languages. Thankfully, users of Iñupiaq dictionaries are spared many of the headaches that accompany compilation and use of dictionaries for those languages. However, for non-native speakers, the task of identifying and looking up the components of a long Iñupiaq word can still be daunting and time-consuming. The Iñupiaq transducer can help with the word analysis, but it would be an even more powerful tool if coupled with an electronic dictionary. The user would enter a word, and definitions for each stem, postbase, and enclitic in each of the proposed parses would appear, possibly along with an explanation of the meaning conveyed by the word’s inflectional ending. The user could quickly scan these and determine which made the most sense in the word’s context.

For future lexicographic projects, this dictionary-transducer relationship could be taken even further, with the two being developed in tandem. The dictionary would provide the transducer’s lexicon, and the transducer, coupled with a corpus similar to the one used for this project, could provide detailed, ongoing feedback to the lexicographer on the coverage and accuracy of the current dictionary entries. It could also be used to identify possible examples to include in entries. There is considerable overlap between the process of compiling an adequately descriptive dictionary and the process of creating an accurate lexical transducer, and developing both products together would provide a number of benefits to each.

5.3.3 Spell-Checker

The lower language of a lexical transducer can be viewed as a giant list of words. This happens to be just what is needed for a basic spell checker (Beesley and Karttunen 2003:451). In order to propose corrections, one also needs a way to map strings missing from this list onto the most similar ones in the list (Beesley and Karttunen 2003:451-453). The more or less accepted way to create this mapping is to build a so-called “Levenshtein automaton,” which matches input to any known words within a predetermined edit distance (also called Levenshtein distance or Levenshtein-Damerau distance); Schulz and Mihov (2002) describe the process in detail (see also Pirinen and Lindén 2010). Integrating a Levenshtein automaton into one’s favorite word processor is less well documented, but Per Langgård and Trond Trosterud have successfully created a West Greenlandic spell-checker for Microsoft Word.

With a token recognition rate presently near 80%, it’s worth asking whether the current Iñupiaq transducer is ready to be made into a spell-checker (or conversely, if users will want every fifth word in their documents underlined in squiggly red). Another factor to consider is the lack of semantic controls in the

transducer, which allow it to accept untold numbers of phonologically correct but meaningless strings of morphemes as valid. I must admit that in general, I dislike spell-checkers, particularly when they flag words which I know are correct. However, for beginning students of Iñupiaq, who are less likely to use long, difficult words and more likely to make mistakes, the benefits of such a tool may well outweigh the potential disadvantages.

5.3.3.1 Other Possible Applications

Shinjae Yoo has done some work on Iñupiaq optical character recognition (OCR) (Yoo 2008) and has expressed interest in using the Iñupiaq transducer to improve OCR accuracy. The transducer could also be used as the foundation for software designed to perform higher levels of linguistic analysis, such as lemmatization, part-of-speech tagging, syntactic analysis, and semantic analysis. Such tools, combined with a larger body of Iñupiaq texts, could create interesting possibilities for corpus-based studies of the language.

As was mentioned in the introduction to this thesis, morphological analysis is fundamental to a number of applications, and beneficial for many others (Beesley 2004c; Sarasola 2000). The availability of a lexical transducer for Iñupiaq opens the door to a number of interesting possibilities. I have tried to design it in such a way and to write this thesis in sufficient detail that others could improve and enlarge the transducer's coverage, even without having an in-depth knowledge of computer programming. For those interested in simply using the transducer, either by itself or as a component in other software, it can be treated as a black box and used as documented in Appendices B and C. If used with a proper understanding of its abilities and limitations, it has the potential to simplify a number of tasks and to make many more possible.

Appendix A

Basics of xfst

I do not intend to provide a complete reference for xfst here, but I do hope to provide enough information that a linguist with no experience in regular expressions can understand the rules and other xfst code described in this document. For more information on the xfst language, consult Beesley and Karttunen (2003).

A.1 The xfst Stack

An important concept in xfst is that of “The Stack” (as Beesley and Karttunen [2003] refer to it). The xfst stack is a last-in, first-out data structure for storing finite-state automata and transducers. Finite-state machines can be “pushed” onto (that is, piled onto the top of) or “popped” off (that is, drawn from the top of) the stack. The FSM at the top of the stack is the FSM against which strings may be evaluated.

A.2 Commands and Comments

xfst code consists of commands and human-readable comments. Comments (which xfst ignores) begin with an **exclamation point !** and extend to the end of the line.

Commands consist of a keyword and a set number of arguments. The Iñupiaq transducer xfst code uses the following commands:

source loads and evaluates a file of xfst code, which may define variables and/or manipulate the xfst stack. It takes one argument, the name of the file to be loaded.

read lexc loads a file of lexc code, compiles the FSM described by that code, and pushes that FSM onto the stack. It takes one argument, the name of the file to be loaded.

read regex compiles an xfst regular expression and pushes the resulting FSM onto the xfst stack. It takes one argument, an xfst expression (see A.3 on pages 169–170).

save stack creates a binary file containing the current contents of the xfst stack. It takes one argument, the name of the file to be created or overwritten.

define sets or resets variables. It has two forms. The first form requires two arguments: a variable name and an expression (see A.3 on pages 169–170). The variable will then store the FSM denoted by the expression. The second form requires just one argument, a variable name. This form pops the top FSM off the stack and stores it in the variable.

eliminate flag converts the FSM on top of the xfst stack to an equivalent FSM not containing a particular flag diacritic (see 2.3.5 on pages 55–58). It takes one argument, the name of the diacritic to be removed from the FSM.

push pushes an FSM contained in a variable onto the xfst stack. It takes one argument, a variable name.

pop pops the topmost FSM off the xfst stack.

substitute defined replaces a symbol in the FSM on top of the xfst stack with an FSM defined in a variable. It takes three arguments: the name of the variable containing the FSM that will replace the

symbol, the keyword “for,” and the symbol to be replaced.

A.3 Regular Expressions in xfst

Regular expressions are made up of symbols, variables, operators, and other syntactic characters (such as square brackets and spaces).

[Square brackets] group together regular expression code much like parentheses group mathematical operations in algebraic notation.

Space is mandatory between two distinct symbols, between two variable names, and between a symbol and a variable name. Thus, the expressions `[string]` and `[s t r i n g]` describe distinct FSAs; the first matches a single, multicharacter symbol (or the contents of a variable named “string,” if such a variable exists), while the second matches a sequence of six single-character symbols. Space is optional elsewhere; for example, the expressions `[s t r i n g]` and `[s t r i n g]` are equivalent.

Many non-alphabetic characters have special meaning in xfst. To use these characters as symbols (or to use them within multi-character symbols) requires special syntax. A **percent sign %** before a character indicates that the character should be interpreted literally rather than as an xfst operator or syntactic character. **Double quotes ""** around a string indicate that the entire string should be treated as a single symbol, with any special characters inside the quotes treated literally. The expressions `[%P.VALENCE.INTR%]` and `["@P.VALENCE.INTR"]` are equivalent.

The **question mark ?** is a wildcard representing any symbol.

The string **.#.** denotes the left or right edge of a word.

Zero 0 denotes an empty string. This is useful only in expressions denoting relations (in cases where a pattern on one side of the relation corresponds to an empty string on the other side).

The **dollar sign \$** denotes the language of all strings containing substrings matching the subexpression to its right. The expression `$(c a t]` describes the language of all strings containing the substring `(cat)`, including `(catfish)`, `(catch)`, `(concatenate)`, and `(gbrmgcatxsptw)`, among infinitely many others.

The **tilde ~** is the language complement operator. It denotes the language of all strings not which do not belong to the language defined by the subexpression to its right. For example, the expression `[c a t]` defines a language containing an infinite number of strings, including `(horse)`, `(dog)`, and `(catfish)`, but not `(cat)`. The expression `$(c a t]` defines a language containing an infinite number of strings, none of which contain the substring `(cat)`.

The **backslash ** is the term complement operator. It denotes the set of single symbols not matched by the subexpression to its right. The expression `^m` matches any single symbol except `(m)`.

The **forward slash /** is the ignoring operator. It indicates that the expression to its right be matched any number of times at any point within the expression to its left. Ignoring is discussed in Section 3.3.1 on pages 98-101.

The **asterisk ***, called the Kleene star¹ in the field of regular expressions, indicates that the preceding subexpression should be matched zero or more times. The pattern `^<n` indicates that the preceding subexpression should be matched zero to $n-1$ times. The pattern `^{m,n}` indicates that the preceding

¹Named after American mathematician Stephen C. Kleene, considered the inventor of regular expressions, who pronounced his last name /kleemi:/.

subexpression should be matched m to n times. Parentheses **()** around a subexpression indicate that the subexpression may optionally be matched (in other words, should be matched either zero times or once).

The **pipe** character **|** is the union or disjunction operator, and can be thought of as corresponding to the conjunction 'or'; for example, the expression $[x | y]$ matches either the character $\langle x \rangle$ or the character $\langle y \rangle$.

The **hyphen** or **minus sign** **-** is the subtraction operator. It calculates the set difference (also called the relative complement) of two subexpressions denoting languages; in other words, it describes a language containing all members of the first language which are not also members of the second language.

The string **.o.** is the composition operator. This operator defines relations such that the upper-side language of the expression to the left of the operator is mapped onto the lower-side language of the expression to the right of the operator. Composition is explained in greater detail on page 43.

The string **->** is the left replacement operator. It takes two subexpressions as operands, one on each side of the operator, and defines a relation such that each string in the upper language is mapped onto an identical string in the lower language, except in the case of an upper-language string containing a substring matched by the left subexpression; such a string is mapped onto the set of strings which are identical to the upper-language string except that the matched substring is replaced with a substring defined by the right subexpression. For example, the expression $[e \rightarrow i]$ defines a relation where every $\langle e \rangle$ in the upper language corresponds to an $\langle i \rangle$ in the lower language. It's important to note that the constraint imposed by the left replacement operator is directional: in the example above, $\langle i \rangle$ in the lower language corresponds not only to $\langle e \rangle$ in the upper language, but also to $\langle i \rangle$ in the upper language. The right replacement operator **<-** constrains relations in the opposite direction. Parentheses around a replacement operator make replacement optional. Replacement operations can be conditioned on a particular surrounding environment using the context separator **||** followed by a subexpression denoting the preceding environment, an underscore **_** and a subexpression denoting the following environment. Either environment subexpression may be omitted. The expression $[n \rightarrow \tilde{n} || i _]$ is an example of conditional replacement; it specifies a relation where strings in the upper language which contain $\langle in \rangle$ correspond to identical strings in the lower language except with $\langle i\tilde{n} \rangle$ in place of $\langle in \rangle$. All other strings in the upper language, including strings which contain $\langle n \rangle$ word-initially or following any symbol other than $\langle i \rangle$, correspond to identical strings in the lower language. It is possible to specify that multiple replacement operations should happen in the same context. This is done by listing all replacement operations before the context separator; for example, $[n \rightarrow \tilde{n}, l \rightarrow |, \dagger \rightarrow \ddagger || i _]$ specifies three mappings that should take place following the symbol $\langle i \rangle$. Finally, it is possible to specify a relation with multiple replacement operations, each sensitive to a different conditioning environment; this is done by putting a double comma **,,** between each replacement operation expression. For example, the expression $[k \rightarrow g || _ l, k \rightarrow \eta || _ n]$ specifies that $\langle k \rangle$ in the upper language correspond to $\langle g \rangle$ in the lower language when followed by $\langle l \rangle$, and to $\langle \eta \rangle$ in the lower language when followed by $\langle n \rangle$.

Appendix B

Grammar Tags Used in the Iñupiaq Transducer

Inflectional endings in the transducer are represented with strings of “grammar tags”—special multicharacter symbols beginning with a plus sign and indicating some grammatical property of the inflectional ending, such as syntactic category, mood, or case. This appendix is divided into two parts: a glossary of tags and a grammar specifying what combinations of tags are possible.

B.1 Glossary of Grammar Tags

Note that “reflexive” here means “coreferent with the subject of the matrix verb.” Endings labeled “reflexive” below are never used to communicate that the same person or entity is both subject and direct object of the same transitive verb. To express this idea in Iñupiaq, the transitive verb is inflected with an intransitive ending.

+1Du	1st person dual subject, possessor, or referent
+1DuO	1st person dual direct object
+1Pl	1st person plural subject, possessor, or referent
+1PlO	1st person plural direct object
+1Sg	1st person singular
+1SgO	1st person singular direct object
+2Du	2nd person dual subject, possessor, or referent
+2DuO	2nd person dual direct object
+2Pl	2nd person plural subject, possessor, or referent
+2PlO	2nd person plural direct object
+2Sg	2nd person singular subject, possessor, or referent
+2SgO	2nd person singular direct object
+3Du	3rd person dual subject, possessor, or referent
+3DuO	3rd person dual direct object
+3Pl	3rd person plural subject, possessor, or referent
+3PlO	3rd person plural direct object
+3RDu	3rd person reflexive dual subject, possessor, or referent
+3RDuO	3rd person reflexive dual direct object
+3RPl	3rd person reflexive plural subject, possessor, or referent
+3RPlO	3rd person reflexive plural direct object
+3RSg	3rd person reflexive singular subject, possessor, or referent
+3RSgO	3rd person reflexive singular direct object
+3Sg	3rd person singular subject, possessor, or referent
+3SgO	3rd person singular direct object
+Abl	ablative case
+Abs	absolutive case

+Adv	adverb (grammatical category)
+Cond	conditional mood (alternatively, consequential-conditional mood, unrealized aspect)
+Conj	conjunction (grammatical category)
+Conseq	consequential mood (alternatively, consequential-conditional mood, realized aspect)
+Cont1	contemporative 1 mood
+Cont2	contemporative 2 mood
+ContNeg	contemporative negative mood
+Dem	demonstrative (grammatical category)
+Du	dual number
+Gerund	gerundive form
+Imp	imperative mood
+ImpNeg	imperative negative mood
+Ind	indicative mood
+Int	interrogative mood
+Interj	interjection (grammatical category)
+Kiis	“kiisaimmaa” mood
+Loc	locative case
+Mod	modalis case
+N	noun (grammatical category)
+Opt	optative mood
+Part	participial form
+Pl	plural number
+Pro	pronoun (grammatical category)
+Prs	“present tense”
+Pst	“past tense”
+Real	realized aspect
+Rel	relative case
+Sg	singular number
+Sim	similaris case
+Simul1	simultaneitive 1 mood
+Simul2	simultaneitive 2 mood
+Simul3	simultaneitive 3 mood
+Trm	terminalis case
+Unreal	unrealized aspect
+V	verb (grammatical category)
+Via	vialis case
+Voc	vocative “case”

B.2 Allowable Grammar Tag String Combinations

B.2.1 Verbs

B.2.1.1 Indicative Mood

The tags in the last column are direct objects and should be omitted for intransitive verbs. Note that sequences of the form +1*+1*O or +2*+2*O (transitive verbs with a 1st or 2nd person subject [any number] and an object of the same grammatical person [any number]) are illegal.

+V	+Ind	+Prs +Pst	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">+1Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+1Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+1Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1PIO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2PIO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3PIO</td><td style="padding: 2px 5px;">)</td></tr> </table>	+1Sg	(+1SgO)	+1Du	(+1DuO)	+1Pl	(+1PIO)	+2Sg	(+2SgO)	+2Du	(+2DuO)	+2Pl	(+2PIO)	+3Sg	(+3SgO)	+3Du	(+3DuO)	+3Pl	(+3PIO)
+1Sg	(+1SgO)																																				
+1Du	(+1DuO)																																				
+1Pl	(+1PIO)																																				
+2Sg	(+2SgO)																																				
+2Du	(+2DuO)																																				
+2Pl	(+2PIO)																																				
+3Sg	(+3SgO)																																				
+3Du	(+3DuO)																																				
+3Pl	(+3PIO)																																				

B.2.1.2 Interrogative Mood

The same subject/object restrictions apply to interrogative verbs as to indicatives.

+V	+Int	<table style="border-collapse: collapse;"> <tr><td style="padding: 2px 5px;">+1Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+1Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+1Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+1PIO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+2Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+2PIO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Sg</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3SgO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Du</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3DuO</td><td style="padding: 2px 5px;">)</td></tr> <tr><td style="padding: 2px 5px;">+3Pl</td><td style="padding: 2px 5px;">(</td><td style="padding: 2px 5px;">+3PIO</td><td style="padding: 2px 5px;">)</td></tr> </table>	+1Sg	(+1SgO)	+1Du	(+1DuO)	+1Pl	(+1PIO)	+2Sg	(+2SgO)	+2Du	(+2DuO)	+2Pl	(+2PIO)	+3Sg	(+3SgO)	+3Du	(+3DuO)	+3Pl	(+3PIO)
+1Sg	(+1SgO)																																			
+1Du	(+1DuO)																																			
+1Pl	(+1PIO)																																			
+2Sg	(+2SgO)																																			
+2Du	(+2DuO)																																			
+2Pl	(+2PIO)																																			
+3Sg	(+3SgO)																																			
+3Du	(+3DuO)																																			
+3Pl	(+3PIO)																																			

B.2.1.3 Contemporative Moods

Note that subjects are not reflected in the conjugation of contemporative transitive verbs.

			+1Sg
			+1Du
			+1Pl
			+2Sg
			+2Du
			+2Pl
			+3Sg
			+3Du
+V	+Cont1	+Real	+3Pl
	+Cont2	+Unreal	+1SgO
			+1DuO
			+1PlO
			+2SgO
			+2DuO
			+2PlO
			+3SgO
			+3DuO
			+3PlO

B.2.1.4 Optative Mood

In the transducer I distinguish between the optative mood, used for 1st and 3rd person subjects, and the imperative mood, used for 2nd person subjects. Same subject/object restrictions apply to optative transitives as apply to indicative transitives.

		+1Sg	+1SgO
		+1Du	+1DuO
		+1Pl	+1PlO
		+3Sg	+2SgO
		+3Du	+2DuO
		+3Pl	+2PlO
+V	+Opt		+3SgO
			+3DuO
			+3PlO

B.2.1.5 Imperative Mood

$$\begin{array}{c}
 +V \quad | \quad +Imp \quad \left| \begin{array}{l} +2Sg \\ +2Du \\ +2Pl \end{array} \right. \quad \left(\begin{array}{l} +1SgO \\ +1DuO \\ +1PlO \\ +3SgO \\ +3DuO \\ +3PlO \end{array} \right)
 \end{array}$$

B.2.1.6 Negative Contemporative Mood

The intransitive negative contemporative serves as the negative of the intransitive optative-imperative. The transitive negative contemporative serves as the negative for transitive imperatives only. The negative of transitive optatives is formed by using the postbase $\pm\eta it-$.

$$\begin{array}{c}
 +V \quad | \quad +ContNeg \quad \left| \begin{array}{l} +1Sg \\ +1Du \\ +1Pl \\ +2Sg \\ +2Du \\ +2Pl \\ +3Sg \\ +3Du \\ +3Pl \end{array} \right. \quad \left(\begin{array}{l} +1SgO \\ +1DuO \\ +1PlO \\ +2SgO \\ +2DuO \\ +2PlO \\ +3SgO \\ +3DuO \\ +3PlO \end{array} \right)
 \end{array}$$

B.2.1.7 Consequential and Conditional Moods

The same subject/object restrictions apply to consequential and conditional transitives as apply to indicative transitives, but note the presence of 3rd person reflexive forms.

+V	+Conseq +Cond	+1Sg	(+1SgO +1DuO +1PlO +2SgO +2DuO +2PlO +3SgO +3DuO +3PlO +3RSgO +3RDuO +3RPlO)
		+1Du	
	+1Pl		
	+2Sg		
	+2Du		
	+2Pl		
	+3Sg		
	+3Du		
	+3Pl		
	+3RSg		
	+3RDu		
	+3RPl		

B.2.2 Nouns

The tags in the last column are possessors and are optional.

+N	+Abs +Rel +Mod +Loc +Trm +Abl +Via +Sim	+Sg +Du +Pl	+1Sg
			+1Du
			+1Pl
			+2Sg
			+2Du
			+2Pl
			+3Sg
			+3Du
			+3Pl
			+3RSg
+3RDu			
+3RPl			

A special set of tags are used for the “vocative” ending $-V\eta$ ‘my dear ____’. Whether it really deserves this treatment is debatable.

+N	+Voc	+Sg	+1Sg
----	------	-----	------

B.2.3 Pronouns

B.2.3.1 Personal Pronouns

Personal pronouns are the only words in the transducer whose upper-language representation consists entirely of grammar tags.

+Pro	+1Sg	+Abs +Rel +Mod +Loc +Trm +Abl +Via +Sim	+Sg +Du +Pl
	+1Du		
	+1Pl		
	+2Sg		
	+2Du		
	+2Pl		
	+3Sg		
	+3Du		
	+3Pl		
	+3RSg		
	+3RDu		
	+3RPl		

B.2.3.2 Kiña, Kisu

Pronouns *kiña* ‘who’ and *kisu* ‘which one’ have no grammatical person tag.

kiña kisu	+Pro	+Sg	+Abs
		+Du	+Rel
		+Pl	+Mod
			+Loc
			+Trm
			+Abl
			+Via
			+Sim

B.2.3.3 Su-

su- ‘what’ may be inflected verbally or (pro)nominally. When used as a verb, its inflection is represented with verbal tags, as outlined above. When used as a pronoun, its inflection is represented using the tags shown below.

su	+Pro	+Abs	+Sg +Du +Pl	+1Sg +1Du +1Pl +2Sg +2Du +2Pl +3Sg +3Du +3Pl +3RSg +3RDu +3RPl
		+Rel		
		+Mod		
		+Loc		
		+Trm		
		+Abl		
		+Via		
		+Sim		

B.2.3.4 Kisi-; Iluqaq-, Tamaq-

kisi- ‘X alone’ and *iluqaq*- and *tamaq*- ‘all of X’ do not fit the ergative pattern of most Iñupiaq nominals. Instead, 1st and 2nd person forms may be used as subjects of transitive and intransitive verbs as well as direct objects of transitive verbs; 3rd person non-reflexive forms may serve as direct objects of transitive verbs; and 3rd person reflexive forms serve as subjects of intransitive and transitive verbs (MacLean n.d.b:ch. 21). Because of this unusual pattern, these pronouns are not marked for case when they act as core arguments of a verb. The only oblique cases for which these pronouns may be inflected are ablative and terminalis (MacLean n.d.b:ch. 21).

kisi iluqaq tamaq	+Pro	+1Sg	+Abl +Trm
		+1Du	
		+1Pl	
		+2Sg	
		+2Du	
		+2Pl	
		+3Sg	
		+3Du	
		+3Pl	
		+3RSg	
+3RDu			
+3RPl			

B.2.4 Demonstratives

B.2.4.1 Demonstrative Adverbs

$$+Dem \mid +Adv \left(\begin{array}{l} +Loc \\ +Via \\ +Abl \\ +Trm \end{array} \right)$$

B.2.4.2 Demonstrative Pronouns

$$+Dem \mid +Pro \left(\begin{array}{l} +Abs \\ +Rel \\ +Mod \\ +Loc \\ +Trm \\ +Abl \\ +Via \\ +Sim \end{array} \right) \left(\begin{array}{l} +Sg \\ +Du \\ +Pl \end{array} \right)$$

B.2.5 Adverbs, Conjunctions, Interjections

Adverbs, conjunctions, and interjections are not paradigmatic; they are marked with a single grammatical category tag which, unlike the other grammar tags used in the transducer, does not correspond to any inflectional ending.

+Adv
+Conj
+Interj

Appendix C

Files, Dependencies, Build Process, and Use of the Transducer

This appendix documents the files associated with the transducer, the dependencies which must be satisfied in order to build the transducer, the steps that must be taken to build the transducer, and two ways to use the transducer once it is built.

C.1 Files

The root directory of the transducer contains four subdirectories: `scripts`, `data`, `intermediate`, and `xfst`. The contents of these directories are as follows:

`scripts/`

<code>run.tcl</code>	Tcl script to convert data files to lexc and xfst formats (calls <code>generate_inflection.tcl</code> , <code>lexicon_to_xml.tcl</code> , and <code>xml_to_lexc.tcl</code>)
<code>generate_inflection.tcl</code>	Tcl script to convert <code>data/inflections.txt</code> to XML format (see Section 3.6 on pages 108–110)
<code>lexicon_to_xml.tcl</code>	Tcl script to convert remaining lexical data to XML format (see Section 3.6 on pages 108–110)
<code>xml_to_lexc.tcl</code>	Tcl script to convert XML-formatted lexicon to lexc and xfst formats (see Section 3.7 on pages 110–114)

`data/`

<code>stems.txt</code>	stem data for lexicon (see Section 3.2.1 on pages 70–77)
<code>postbases.txt</code>	postbase data for lexicon (see Section 3.2.2 on pages 77–83)
<code>inflections.txt</code>	inflectional data for lexicon (see Section 3.2.3 on pages 83–95)
<code>enclitics.txt</code>	enclitic data for lexicon (see Section 3.2.4 on pages 95–96)
<code>classes.txt</code>	formative class declarations, including definitions of epsilon continuations (see Section 3.4 on pages 104–105)
<code>multichar_symbols.txt</code>	declarations of multicharacter symbols (see Section 3.5 on pages 105–106)
<code>patterns.txt</code>	definitions of allomorph conditioning environments (see Section 3.3 on pages 96–104)

`intermediate/`

<code>inflections.xml</code>	inflectional endings in XML format; generated by <code>generate_inflection.tcl</code>
<code>lexicon.xml</code>	lexicon in XML format; generated by <code>lexicon_to_xml.tcl</code>

`xfst/`

<i>ipk_xfst.txt</i>	morphographemic rules and instructions to build the transducer (see Section 3.8 114–145)
<i>ipk_lexc.txt</i>	lexicon in lexc format; generated by <i>xml_to_lexc.txt</i>
<i>ipk_condition_xfst.txt</i>	allomorph conditioning environment filters; generated by <i>xml_to_lexc.txt</i>
<i>ipk.fst</i>	non-guessing transducer; generated by <i>ipk_xfst.txt</i>
<i>ipk_guesser.fst</i>	guessing transducer; generated by <i>ipk_xfst.txt</i>

C.2 Dependencies

The following software is required to build the transducer:

- Tcl 8.5 or greater (<http://www.tcl.tk/>); note: the Tk library is not required
- tDOM 0.8.2 or greater (<http://wiki.tcl.tk/tDOM>)
- struct::set 2.2.3 or greater (http://tcllib.sourceforge.net/doc/struct_set.html)
- XFST 2.14.1 or greater (<http://www.fsmbook.com>)

C.2.1 Tcl

Tcl may be obtained in a number of ways. For purposes of building the Iñupiaq transducer, I recommend one of the following:

- Many Linux distributions include Tcl in their software repositories; consult your distribution’s documentation for more details.
- ActiveState Corporation provides a convenient Tcl distribution called ActiveTcl; see <http://www.activestate.com/activetcl> (look for the “Free Community Edition”).
- Evolane distributes a “batteries-included” Tcl distribution called eTcl, which can be downloaded at <http://www.evolane.com/software/etcl/>.

In addition to installing Tcl, you should make sure that the directory containing the Tcl binaries (usually called bin, e.g., *etcl/bin*, *ActiveTcl-8.5/bin*) is listed in your PATH environment variable.

C.2.2 tDOM, struct::set

tDOM is a Tcl extension used for creating, parsing, and modifying XML documents using the Document Object Model. Its primary authors are Jochen Löwer and Rolf Ade; Zoran Vasiljevic has also made notable contributions.

struct::set is a Tcl extension for performing set-theoretic operations. It is part of the tcllib collection of Tcl extensions. struct::set was written by Andreas Kupries.

How one obtains these packages depends upon the Tcl distribution being used. eTcl comes with tDOM and tcllib, so eTcl users don’t need to do anything. Linux users may be able to download tDOM and tcllib using their distribution’s software repository. ActiveTcl users can get these from the “Teapot” repository by typing the following at the command line (may require administrator privileges):

```
teacup install tdom 0.8.3
teacup install struct::set 2.2.3
```

C.2.3 XFST

At the time of writing, XFST is available for non-commercial use at <http://www.fsmbook.com>, subject to terms which must be agreed to as part of the download process. The transducer was compiled using xfst version 2.12.12 and lexc version 3.7.9 (both with libcfsm-2.17.9).

As with Tcl, ensure that the directory containing the xfst executables is listed in your PATH environment variable.

C.3 Build Process

The build process is carried out in two parts: running `run.tcl` and running `ipk_xfst.txt`. Both parts are accomplished on the command line.

STEP 1 Ensure that the transducer files are saved in a location where you have write access, such as your hard drive or a USB flash drive.

STEP 2 Gain access to your command line. On Windows, this is done by going to the Start Menu and selecting All Programs, then Accessories, then Command Prompt. On OS X, open the Applications folder, open Utilities, and open Terminal. I assume that Linux and Unix users know how to access the command line. All remaining steps will take place on the command line.

STEP 3 Use the `cd` command to change directories to the `scripts/` directory. For example, if you are using Windows and the transducer files are located in `C:\inupiaqtransducer\`, you would type

```
C:
cd C:\inupiaqtransducer\scripts
```

If you are using OS X, and the transducer files are in `/Users/Alice/inupiaqtransducer/`, you would type

```
cd /Users/Alice/inupiaqtransducer/scripts
```

STEP 4 Invoke the script `run.tcl` by typing `<name of Tcl executable> run.tcl`. The name of the Tcl executable will probably be something like `tclsh`, `tclsh85`, `tclsh8.5`, or `etcl`. If in doubt, look in the `bin` directory of your Tcl distribution. The script `run.tcl` should complete in under a minute.

STEP 5 Change the working directory to the `xfst` directory. On Windows, this is done by typing

```
cd ..\xfst
```

On OS X, Linux, and Unix, it is done by typing

```
cd ../xfst
```

STEP 6 Invoke the `xfst` script `ipk_xfst.txt` by one of the methods outlined below. Just do one or the other, not both. Regardless of the option chosen, this step will take a long time, perhaps as long as ten minutes.

OPTION 6A If you just want to build the transducer binaries and don't want to interact with the transducer afterward, type

```
xfst -f ipk_xfst.txt
```

OPTION 6B Alternatively, if you want to interact with the transducer following compilation, type `xfst` at the command prompt; then, at the XFST prompt, type

```
source ipk_xfst.txt
```

See the following section for instructions on interacting with the transducer.

C.4 Using the Compiled Transducer

There are at least two ways to use the transducer: from within XFST, or with the lookup utility which comes with XFST.

C.4.1 Within XFST

If you just compiled the transducer and used Option 6B for Step 6, the non-guessing transducer will already be on top of the XFST stack; jump to Step 5 below. Otherwise, begin with Step 1.

STEP 1 Gain access to your command line as outlined in Step 2 of the build instructions.

STEP 2 Use the `cd` command to change directories to the `scripts/` directory. For example, if you are using Windows and the transducer files are located in `C:\inupiaqtransducer\`, you would type

```
C:
cd C:\inupiaqtransducer\scripts
```

If you are using OS X, and the transducer files are in `/Users/Alice/inupiaqtransducer/`, you would type

```
cd /Users/Alice/inupiaqtransducer/scripts
```

STEP 3 Run `xfst`. Type `xfst` at the command prompt.

STEP 4 Load the non-guessing transducer by typing

```
load ipk.fst
```

Alternatively, if you wish to interact with the guessing transducer instead, skip this step and go to Step 5.

STEP 5 If you want to interact with the guessing transducer rather than the non-guessing transducer, type

```
load ipk_guesser.txt
```

At this point, you can interact with the transducer in a number of ways. Since there is no inherent order that need be followed for these steps, I will name them rather than number them.

ANALYZING A SINGLE WORD Type apply up <word> or simply up <word>. For example, to analyze the word *nakuuruᅇa*, type

```
up nakuuruᅇa
```

SYNTHESIZING A SINGLE WORD Type apply down <morpheme string> or simply down <morpheme string>. For example, to synthesize a word from the string *uqauti>rraqsi>+V+Ind+Prs+3Sg+3PIO*, type

```
down uqauti>rraqsi>+V+Ind+Prs+3Sg+3PIO
```

ANALYZING OR SYNTHESIZING SEVERAL WORDS Type apply up or up to enter analysis mode, or apply down or down to enter synthesis mode. The prompt will change to a greater-than sign. You may now enter Inupiaq words (in analysis mode) or morpheme strings (in synthesis mode) directly at the prompt. When you are done and wish to exit the special mode, type

```
END;
```

EXITING XFST To quit the program, type

```
exit
```

C.4.2 Using the lookup Utility

The lookup utility is a powerful tool for analyzing several words in a single batch. Much more could be said about this tool than I will say here; for more information, consult Beesley and Karttunen (2003) and <http://www.fsmbook.com>.

STEP 1 To use this tool, you will need to prepare a file containing the words to be analyzed, with one word per line. The most convenient place to save this file is probably in the *xfst/* directory, but you can save it wherever you like.

STEP 2 Gain access to your command line as outlined in Step 2 of the build instructions.

STEP 3 Change the working directory to the *xfst* directory within the transducer files. For example, if you are using Windows and the transducer files are stored in *C:\inupiaqtransducer*, you would type

```
c:
cd c:\inupiaqtransducer\xfst
```

If you are using OS X and the transducer files are stored in */Users/Alice/inupiaqtransducer/*, you would type

```
cd /Users/Alice/inupiaqtransducer/xfst
```

STEP 4 The way you invoke lookup depends on what you want to do with it, but also with the operating system you are using. An example of a basic call to lookup might look something like this:

```
<words.txt lookup ipk.fst -flags L:LTT -utf8
```


Here, the input file (containing one word per line) is `words.txt`, which is located in the `xfst/` directory. The transducer to be applied is `ipk.fst`, the non-guessing transducer. Results are displayed directly on screen. On Windows, this could also be accomplished like this:

```
type words.txt | lookup ipk.fst -flags L:LTT -utf8
```

The Unix/Linux/OS X version of this is

```
cat words.txt | lookup ipk.fst -flags L:LTT -utf8
```

The `cat/type` method has the advantage that you are not likely to type a greater-than sign rather than a less-than sign (an accidental greater-than sign would delete your input file).

To save the result to a file (say, `lookupoutput.txt`), you would type:

```
<words.txt lookup ipk.fst -flags L:LTT -utf8 >lookupoutput.txt
```

(As with the previous example, you can replace the less-than sign with a call to `type` or `cat` [depending on your operating system] plus a pipe following the input file name.)

It is also possible to do complex lookups involving more than one transducer. Instructions on how to perform the lookups can be specified in a “lookup strategy script.” To attempt an analysis of a word by first looking in the lexical transducer, then in the guessing transducer, one could use a script like the following:

```
nonguessing ipk.fst
guessing ipk_guesser.fst

nonguessing
guessing
```

Assuming the data above were stored in the file `lookupsript.txt` in the `xfst/` directory, you could use this script by typing

```
<words.txt lookup -f lookupsript.txt -flags L:LTT -utf8
```

To save the results to a file (say, `lookupoutput.txt`), you would type

```
<words.txt lookup -f lookupsript.txt -flags L:LTT -utf8
```

Bibliography

All URLs were valid as of April 6, 2011.

- Aduriz, Itziar, Iñaki Alegria, Jose Mari Arriola, Xabier Artola, Arantza Díaz de Ilarraza, Nerea Ezeiza, Koldo Gojenola, and Montserrat Maritxalar. 1995. Different issues in the design of a lemmatizer/tagger for Basque. In Mike Armstrong and Evelyne Tzoukermann, editors, *Proceedings of the EACL-SIGDAT Workshop 'From text to tags: issues in multilingual text analysis', March 27, Dublin, Ireland*.
<http://arxiv.org/abs/cmp-lg/9503020>
- Aho, Alfred V. and Jeffrey D. Ullman. 1995. *Foundations of Computer Science: C Edition*. Computer Science Press, New York, New York.
- Ahvakana, Floyd. 1973. *Tikiǵaǵmigguuq.... = In Point Hope, Alaska....* Alaska Native Language Center, Fairbanks, Alaska.
- Ahvakana, Floyd. 1975. *Avaqqanam Quliaqtuaqtanik: I. Qupquǵiaq II. Iñuqulligaurat = I. The Ten-Legged Polar Bear II. Dwarves*. Alaska Native Language Center, Fairbanks, Alaska.
- Alegria, Iñaki, Xabier Artola, Kepa Sarasola, and Miriam Urkia. 1996. Automatic morphological analysis of Basque. *Literary and Linguistic Computing*, 11(4):193-203. A pre-publication version is available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.2375&rep=rep1&type=pdf>. The present work refers to the published version.
- Anderson, Stephen R. 1988. Morphology as a parsing problem. In Karen Wallace, editor, *Morphology as a Computational Problem*, UCLA Occasional Papers 7: Working Papers in Morphology. UCLA Department of Linguistics, Los Angeles, California.
- Antonsen, Lene, Saara Huhmarniemi, and Trond Trosterud. 2009. Interactive pedagogical programs based on constraint grammar. In Kristiina Jokinen and Eckhard Bick, editors, *Proceedings of the 17th Nordic Conference of Computational Linguistics NODALIDA 2009*, pages 10-17. Tartu University Library, Tartu, Estonia.
<http://hdl.handle.net/10062/9546>
- Atkins, B. T. S. 1996. Bilingual dictionaries: Past, present and future. In Martin Gellerstam, Jerker Järborg, Sven-Göran Malmgren, Kerstin Noren, Lena Rogström, and Catarina Røjder Pappmehl, editors, *Proceedings of the EURALEX 1996 International Congress (EURALEX '96), August 13-18, Göteborg, Sweden*, pages 515-546. University of Gothenburg Department of Swedish, Göteborg, Sweden.
- Beesley, Kenneth R. 1998. Constraining separated morphotactic dependencies in finite-state grammars. In Lauri Karttunen and Kemal Oflazer, editors, *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, June 30-July 1, Ankara, Turkey*, pages 118-127.
<http://www.aclweb.org/anthology-new/W/W98/W98-1312.pdf>
- Beesley, Kenneth R. 2003. Finite-state morphological analysis and generation for Aymara. In *Proceedings of the Workshop on Finite State Methods in Natural Language Processing, 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 10), April 13-14 2003, Budapest, Hungary*, pages 19-26.
http://www.cl.cam.ac.uk/~ar283/eacl03/workshops03/W03-w9_eacl03beesley.local.pdf
- Beesley, Kenneth R. 2004a. Downtranslation of XML dictionaries to lexc LEXICONS: Third draft. Published online: <http://www.stanford.edu/~laurik/fsmbook/clarifications/xmldowntrans.html>.

- Beesley, Kenneth R. 2004b. The lookup utility and lookup-strategy scripts. Published online: <http://www.stanford.edu/~laurik/fsmbook/clarifications/lookup.html>.
- Beesley, Kenneth R. 2004c. Morphological analysis and generation: A first step in natural language processing. In Julie Carson-Berndsen, editor, *First Steps in Language Documentation for Minority Languages: Computational Linguistic Tools for Morphology, Lexicon and Corpus Compilation, Proceedings of the SALTMIL Workshop at LREC 2004, May 24, Lisbon, Portugal*, pages 1-8.
<http://gandalf.aksis.uib.no/non/lrec2004/ws/ws2.pdf#page=7>
- Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI, Stanford, California.
- Bescherelle. 1995. *Complete Guide to Conjugating 12000 French Verbs*. Hatier, Paris, France.
- Bills, Aric, Lori S. Levin, Lawrence D. Kaplan, and Edna Ahgeak MacLean. 2010. Finite-state morphology for Iñupiaq. In Kefa Sarasola, Francis M. Tyers, and Mikel L. Forcada, editors, *7th SaLTMiL Workshop on Creation and Use of Basic Lexical Resources for Less-Resourced Languages, LREC 2010, Valletta, Malta, 23 May 2010*, pages 19-26.
http://siuc01.si.ehu.es/~jipsagak/SALTMIL2010_Proceedings.pdf
- Black, Alan W., Joke van de Plassche, and Briony Williams. 1991. Analysis of unknown words through morphological decomposition. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL 1991), April 9-11, Berlin, Germany*, pages 101-106. Morgan Kaufmann, San Francisco, California.
<http://www.aclweb.org/anthology-new/E/E91/E91-1018.pdf>
- Blanchett, Marie N. and Martha Teeluk. 1973. *Savaktugut sulii Piuraaqtugut = We Work and We Play*. Alaska Native Language Center, Fairbanks, Alaska.
- Bosch, Sonja E. and Laurette Pretorius. 2003. Building a computational morphological analyzer/generator for Zulu using Xerox Finite-State Tools. In *Proceedings of the Workshop on Finite State Methods in Natural Language Processing, 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 10), April 13-14 2003, Budapest, Hungary*, pages 27-34.
http://www.cl.cam.ac.uk/~ar283/eacl03/workshops03/W03-w9_eacl03bosch.local.pdf
- Carlberger, Johan, Hercules Dalianis, Martin Hassel, and Ola Knutsson. 2001. Improving precision in information retrieval for Swedish using stemming. Technical Report IPLab-194, TRITA-NA-P0116, KTH Department of Numerical Analysis and Computing Science, Stockholm, Sweden.
<ftp://ftp.nada.kth.se/IPLab/TechReports/IPLab-194.pdf>
- Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113-124.
<http://www.chomsky.info/articles/195609--.pdf>
- Church, Kenneth W. 2005. The DDI approach to morphology. In Antti Arppe, Lauri Carlson, Krister Lindén, Jussi Piitulainen, Mickael Suominen, Martti Vainio, Hanna Westerlund, and Anssi Yli-Jyrä, editors, *Inquiries into Words, Constraints and Contexts. Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 25-34. CSLI, Stanford, California.
- Compton, Richard and B. Elan Dresher. 2008. Palatalization and 'strong' /i/ across Inuit dialects. In Susie Jones, editor, *Actes du Congrès de l'ACL 2008 = 2008 CLA Conference Proceedings, May 31-June 2, Vancouver, B.C.*
http://homes.chass.utoronto.ca/~cla-acl/actes2008/CLA2008_Compton_Dresher.pdf
- Cornillac, Guy. 2000. Le mot en morceaux : le cas de la langue inuit. In Nicole Tersis and Michèle Therrien, editors, *Les langues eskaléoutes : Sibérie, Alaska, Canada, Groënland*, pages 171-181. CNRS, Paris, France.

- de Reuse, Willem J. 1988. The morphology/semantics interface: An autolexical treatment of Eskimo verbal affix order. In Lynn MacLeod, Gary Larson, and Diane Brentari, editors, *Papers from the 24th Annual Regional Meeting of the Chicago Linguistic Society (CLS 24). Part One: The General Session*, pages 112-125.
- de Reuse, Willem J. 1992. The role of internal syntax in the historical morphology of Eskimo. In Mark Aronoff, editor, *Morphology Now*, pages 163-178. SUNY Press, Albany, New York.
- Denny, J. Peter. 1982. Semantics of the Inuktitut (Eskimo) spatial deictics. *International Journal of American Linguistics*, 48(4):359-384.
- Dorais, Louis-Jacques. 1990. *Inuit Uqausiqatigiit = Inuit Languages and Dialects*. Arctic College—Nunatta Campus, Iqaluit, Nunavut.
- Ekmekçioğlu, F. Çuna and Peter Willett. 2000. Effectiveness of stemming for Turkish text retrieval. *Program*, 34(2):195-200.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.2951&rep=rep1&type=pdf>
- Fortescue, Michael. 1980. Affix ordering in West Greenlandic derivational processes. *International Journal of American Linguistics*, 46(4):259-278.
- Fortescue, Michael. 2004. West Greenlandic (Eskimo). In Geert Booij, Christian Lehmann, Joachim Mugdan, and Stavros Skopeteas, editors, *Morphology: an International Handbook on Inflection and Word-Formation*, volume 2, pages 1389-1399. Walter de Gruyter, Berlin, Germany and New York, New York.
- Galvez, Carmen and Félix de Moya-Anegón. 2006. An evaluation of conflation accuracy using finite-state transducers. *Journal of Documentation*, 62(3):328-349.
<http://www.ugr.es/~cgalvez/Galvez-JDocumentation-1.pdf>
- Galvez, Carmen, Félix de Moya-Anegón, and Victor H. Solana. 2005. Term conflation methods in information retrieval. *Journal of Documentation*, 61(4):520-547.
<http://www.ugr.es/~cgalvez/Galvez-JDocumentation-2.pdf>
- Grimley-Evans, Edmund, George Anton Kiraz, and Stephen G. Pulman. 1996. Compiling a partition-based two-level formalism. In *COLING 1996, 16th International Conference on Computational Linguistics, Proceedings of the Conference, August 5-9, 1996, Copenhagen, Denmark*, pages 454-459. Center for Sprogteknologi, Copenhagen, Denmark.
<http://www.aclweb.org/anthology-new/C/C96/C96-1077.pdf>
- Harman, Donna. 1991. How effective is suffixing? *Journal of the American Society for Information Science*, 42(1):7-15.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9828&rep=rep1&type=pdf>
- Herreid, Ingrid. n.d. *Jennie-m Iñuugigniña Fairbanks-mi = Jennie's Life in Fairbanks*. Alaska Native Language Center, Fairbanks, Alaska.
[http://www.uaf.edu/anla/collections/search/files/IN\(S\)N982H1982/Jennie-m.pdf](http://www.uaf.edu/anla/collections/search/files/IN(S)N982H1982/Jennie-m.pdf)
- Hull, David and Gregory Grefenstette. 1996. A detailed analysis of English stemming algorithms. Technical Report 1996-023, Rank Xerox Research Centre.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.3856&rep=rep1&type=pdf>
- Hussain, Sara. 2004. *Finite-State Morphological Analyzer for Urdu*. Master's thesis, National University of Computer and Emerging Sciences (Pakistan).
http://www.crupl.org/publication/theses/2004/fs_morphological_analyzer_for_urdu.pdf
- Jacobson, Steven A. 1984. *Yup'ik Eskimo Dictionary*. Alaska Native Language Center, Fairbanks, Alaska.

- Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague, Netherlands. Published version of Johnson's 1970 doctoral dissertation.
- Johnson, Christopher R. and Charles J. Fillmore. 2000. The FrameNet tagset for frame-semantic and syntactic coding of predicate-argument structure. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (ANLP-NAACL 2000), April 29-May 4, Seattle, Washington*, pages 56-62. Morgan Kaufmann, San Francisco, California.
http://framenet.icsi.berkeley.edu/papers/crj_cjf2000.pdf
- Kaplan, Lawrence D. 1981a. *North Slope Iñupiaq Literacy Manual*. Alaska Native Language Center, Fairbanks, Alaska.
[http://www.uaf.edu/anla/collections/search/files/IN\(N\)974K1981/North_Slope_Literacy_1981.pdf](http://www.uaf.edu/anla/collections/search/files/IN(N)974K1981/North_Slope_Literacy_1981.pdf)
- Kaplan, Lawrence D. 1981b. On Yupik-Inupiaq correspondences for i: A case of Inupiaq innovation. *Inuit Studies*, 5(Special Issue):81-90.
- Kaplan, Lawrence D. 1981c. *Phonological Issues in North Alaskan Inupiaq*. Number 6 in Research Papers. Alaska Native Language Center, Fairbanks, Alaska. Published version of Kaplan's 1979 doctoral dissertation.
<http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED398760>
- Kaplan, Lawrence D. 1982. Consonant alternation in Inupiaq Eskimo. *International Journal of American Linguistics*, 48(4):385-393.
- Kaplan, Lawrence D. 1990. The language of the Alaskan Inuit. In Dirmid R. F. Collis, editor, *Arctic Languages: An Awakening*, pages 131-158. UNESCO, Paris, France.
<http://unesdoc.unesco.org/images/0008/000861/086162e.pdf>
- Kaplan, Lawrence D. 2000. LInupiaq et les contacts linguistiques en Alaska. In Nicole Tersis and Michèle Therrien, editors, *Les langues eskaléoutes: Sibérie, Alaska, Canada, Groënland*, pages 91-108. CNRS, Paris, France.
- Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331-378.
<http://www.aclweb.org/anthology-new/J/J94/J94-3001.pdf>
- Karlssoon, Fred. 1999. *Finnish: An Essential Grammar*. Routledge, London, England.
- Karttunen, Lauri. 1994. Constructing lexical transducers. In *COLING 1994, 15th International Conference on Computational Linguistics, August 5-9, Kyoto, Japan*, volume 1, pages 406-411.
<http://acl.ldc.upenn.edu/C/C94/C94-1066.pdf>
- Karttunen, Lauri. 1997. The replace operator. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 117-147. MIT Press, Cambridge, Massachusetts.
- Karttunen, Lauri and Kenneth R. Beesley. 1992. Two-level rule compiler. Technical Report ISTL-92-2, Xerox Palo Alto Research Center, Palo Alto, California.
<http://www.cis.upenn.edu/~cis639/docs/twolc.html>
- Karttunen, Lauri and Kenneth R. Beesley. 2005. Twenty-five years of finite-state morphology. In Antti Arppe, Lauri Carlson, Krister Lindén, Jussi Piitulainen, Mickael Suominen, Martti Vainio, Hanna West-erlund, and Anssi Yli-Jyrä, editors, *Inquiries into Words, Constraints and Contexts. Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 71-83. CSLI, Stanford, California.

- Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *COLING 1992, 14th International Conference on Computational Linguistics, August 23-28, Nantes, France*, pages 141-148.
<http://acl.ldc.upenn.edu/C/C92/C92-1025.pdf>
- Kaveolook, Harold. 1974. *Malguk Quliaqtuak: Aahaalliglu Piyaaniglu / Aniqpaktuaq Avinngaq = Two Stories: The Old Squaw and Its Ducklings / The Large Lemming*. Alaska Native Language Center, Fairbanks, Alaska.
- Kettunen, Kimmo, Tuomas Kunttu, and Kalervo Järvelin. 2005. To stem or lemmatize a highly inflectional language in a probabilistic IR environment? *Journal of Documentation*, 61(4):476-496. A pre-publication draft is available online: http://www.info.uta.fi/tutkimus/fire/archive/kettunen_et_al_full_version_2005.pdf. The present work refers to the published version.
- Khaltar, Badam-Osor and Atsushi Fujii. 2008. A lemmatization method for modern Mongolian and its application to information retrieval. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008), January 7-12, Hyderabad, India*.
<http://www.aclweb.org/anthology-new/I/I08/I08-1000.pdf>
- Kiraz, George Anton. 2000. Multitiered nonlinear morphology using multitape finite automata: A case study on Syriac and Arabic. *Computational Linguistics*, 26(1):77-105.
<http://www.mitpressjournals.org/doi/pdf/10.1162/089120100561647>
- Kiraz, George Anton. 2001. *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge University Press, Cambridge, England and New York, New York.
- Korenien, Tuomo, Jorma Laurikkala, Kalervo Järvelin, and Martti Juhola. 2004. Stemming and lemmatization in the clustering of Finnish text documents. In *Proceedings of the ACM Thirteenth Conference on Information and Knowledge Management (CIKM '04), November 8-13, Washington, D.C.*
<http://www.info.uta.fi/tutkimus/fire/archive/KLJJ-CIKM04.pdf>
- Kornai, András. 1999. Extended finite state models of language. In András Kornai, editor, *Extended Finite State Models of Language*, Studies in Natural Language Processing, pages 1-5. Cambridge University Press, Cambridge, England and New York, New York.
- Koskenniemi, Kimmo. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Ph.D. dissertation, University of Helsinki.
<http://www.ling.helsinki.fi/~koskenni/doc/Two-LevelMorphology.pdf>
- Koskenniemi, Kimmo. 1996. Finite-state morphology and information retrieval. In András Kornai, editor, *Proceedings of the ECAI 96 Workshop on Extended Finite State Models of Language, August 11-12, Budapest, Hungary*, pages 42-45. NJSZT, Budapest, Hungary.
<http://kornai.com/EFS/OnlineSupportMaterial/ECAI/kosk.pdf>
- Krauss, Michael E. 2007. Native languages of Alaska. In Osahito Miyaoka, Osamu Sakiyama, and Michael E. Krauss, editors, *Vanishing Voices of the Pacific Rim*, pages 406-417. Oxford University Press, New York, New York and Oxford, England.
- Krovetz, Robert. 1993. Viewing morphology as an inference process. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett, editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, June 27-July 1, Pittsburgh, Pennsylvania*, pages 191-202. Association for Computing Machinery, New York, New York.
<ftp://ftp.cs.umass.edu/pub/techrept/techreport/1993/UM-CS-1993-036.ps>

- Langgård, Per and Trond Trosterud. n.d. *Iñupiaq parser project*. Computer software. The project home-page is <http://giellatekno.uit.no/ipk.html>. Source code is available for online browsing or anonymous Subversion check-out from <https://victorio.uit.no/langtech/trunk/st/ipk/>.
- Leslau, Wolf. 1995. *Reference Grammar of Amharic*. Harrassowitz, Wiesbaden, Germany.
- Lowe, Ronald. 1996. Grammatical sketches: Inuktitut. In Jacques Maurais, editor, *Quebec's Aboriginal Languages: History, Planning, and Development*, pages 240–232. Multilingual Matters, Philadelphia, Pennsylvania.
- Lowe, Ronald. 2000. Systématique du mot inuit. In Nicole Tersis and Michèle Therrien, editors, *Les langues eskaléoutes : Sibérie, Alaska, Canada, Groënland*, pages 149–170. CNRS, Paris, France.
- MacLean, Edna Ahgeak. 1981. *Iñupiallu Tanñiġlu Uqaluġisa Iġaŋich = Abridged Iñupiaq and English Dictionary*. Alaska Native Language Center, Fairbanks, Alaska.
- MacLean, Edna Ahgeak. 1985. *North Slope Iñupiaq Dialogues: Supplement to North Slope Iñupiaq Grammar: First Year*. Alaska Native Language Center, Fairbanks, Alaska.
- MacLean, Edna Ahgeak. 1986a. *North Slope Iñupiaq Grammar: First Year*. 3rd edition. Alaska Native Language Center, Fairbanks, Alaska.
- MacLean, Edna Ahgeak. 1986b. *North Slope Iñupiaq Grammar: Second Year (Preliminary Edition for Student Use Only)*. Alaska Native Language Center, Fairbanks, Alaska.
- MacLean, Edna Ahgeak. 1986c. *Quliaqtuat Mumiaksrat: Iġisaqtuanun Savaaksriat*. Alaska Native Language Center, Fairbanks, Alaska.
- MacLean, Edna Ahgeak. n.d.a. *North Slope Iñupiaq Dictionary*. Unpublished manuscript (July 14, 2009 revision).
- MacLean, Edna Ahgeak. n.d.b. *North Slope Iñupiaq Grammar: Third Year*. Unpublished manuscript.
- Manning, Christopher D. and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts.
- Mather, Elsie. 1973. *Iqiasuaq Aviŋŋaq = The Lazy Mouse*. Alaska Native Language Center, Fairbanks, Alaska.
- Matoušek, Jiří and Jaroslav Nešetřil. 2009. *Invitation to Discrete Mathematics*. 2nd edition. Oxford University Press, New York, New York and Oxford, England.
- Maxwell, Michael and William Poser. 2004. Morphological interfaces to dictionaries. In Michael Zock and Patrick Saint Dizier, editors, *COLING 2004, 20th International Conference on Computational Linguistics, Proceedings of the Workshop on Enhancing and Using Electronic Dictionaries*, pages 65–68.
- Mithun, Marianne. 2000. Valency-changing derivation in Central Alaskan Yup'ik. In R. M. W. Dixon and Alexandra Y. Aikhenvald, editors, *Changing Valency: Case Studies in Transitivity*, pages 84–114. Cambridge University Press, Cambridge, England and New York, New York.
- Miyaoka, Osahito. 2000. Morphologie verbale en yupik alaskien central. In Nicole Tersis and Michèle Therrien, editors, *Les langues eskaléoutes : Sibérie, Alaska, Canada, Groënland*, pages 225–248. CNRS, Paris, France.
- Mohri, Mehryar. 1997. On the use of sequential transducers in natural language processing. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 355–382. MIT Press, Cambridge, Massachusetts.

- Nageak, Vincent. 1975. *Ataatalugiik = Grandchild with Grandfather*. Alaska Native Language Center, Fairbanks, Alaska.
- Nashaknik, Henry. 1973. *Aḡuḡhuyuk*. Alaska Native Language Center, Fairbanks, Alaska.
- Ousterhout, John K. and Ken Jones. 2009. *Tcl and the Tk Toolkit*. 2nd edition. Addison-Wesley, Reading, Massachusetts.
- Pirinen, Tommi A. and Krister Lindén. 2010. Finite-state spell-checking with weighted language and error models—building and evaluating spell-checkers with wikipedia as corpus. In Kepa Sarasola, Francis M. Tyers, and Mikel L. Forcada, editors, *7th SaLTMiL Workshop on Creation and Use of Basic Lexical Resources for Less-Resourced Languages, LREC 2010, Valletta, Malta, 23 May 2010*, pages 13-18. http://siuc01.si.ehu.es/~jipsagak/SALTMIL2010_Proceedings.pdf
- Pirkola, Ari. 2001. Morphological typology of languages for IR. *Journal of Documentation*, 57(3):330-348. A pre-publication draft is available: www.info.uta.fi/tutkimus/fire/archive/morphological_typology.pdf. The present work refers to the published version.
- Porter, Martin F. 1980. An algorithm for suffix stripping. *Program*, 14(3):130-137.
- Poser, William. 2002. Making Athabaskan dictionaries usable. In Siri G. Tuttle and Gary Holton, editors, *Proceedings of the Athabaskan Languages Conference*, pages 136-147. Alaska Native Language Center, Fairbanks, Alaska. <http://billposer.org/Papers/makath.pdf>
- Roark, Brian and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press, New York, New York and Oxford, England.
- Roche, Emmanuel and Yves Schabes. 1997. Introduction. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 1-66. MIT Press, Cambridge, Massachusetts.
- Ruessink, Herbert. 1989. Two-level formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 5.
- Sarasola, Kepa. 2000. Strategic priorities for the development of language technology in minority languages. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000), May 31-June 2, Athens, Greece*. European Language Resources Association, Paris, France. <http://ixa.si.ehu.es/Ixa/Argitalpenak/Artikuluak/1000911718/publikoak/00LRECSarasola.ps>
- Savoy, Jacques. 1999. A stemming procedure and stopword list for general French corpora. *Journal of the American Society for Information Science*, 50(10):944-952. A pre-publication draft is available: <http://members.unine.ch/jacques.savoy/Papers/FRJasis.pdf>. The present work references the published version.
- Schulz, Klaus U. and Stoyan Mihov. 2002. Fast string correction with Levenshtein-automata. *International Journal on Document Analysis and Recognition*, 5(1):67-85.
- Seiler, Wolf A. 1997. Valence and affix ordering in Inupiatun. In *SIL Electronic Working Papers 1997-002*. Summer Institute of Linguistics. <http://www.sil.org/silewp/1997/002/SILEWP1997-002.html>
- Seiler, Wolf A. 2005. *Inupiatun Eskimo Dictionary*. NANA Regional Corporation, Kotzebue, Alaska.

- Sever, Hayri and Yiltan Bitirim. 2003. Findstem: Analysis and evaluation of a Turkish stemming algorithm. In *String Processing and Information Retrieval*, volume 2857 of *Lecture Notes in Computer Science*, pages 238–251. Springer, Berlin and Heidelberg, Germany. A pre-publication draft is available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.2511&rep=rep1&type=pdf>. The present work refers to the published version.
- Smith, Lawrence R. 1980. Some categories and processes of Labrador Inuttut word formation. *International Journal of American Linguistics*, 46(4):279–288.
- Sproat, Richard. 1992. *Morphology and Computers*. MIT Press, Cambridge, Massachusetts.
- Taff, Alice, Lorna Rozelle, Taehong Cho, Peter Ladefoged, Moses Dirks, and Jacob Wegelin. 2001. Phonetic structures of Aleut. *Journal of Phonetics*, 29:231–271.
http://tcho.hanyang.ac.kr/papers/Taff_Rozelle_Cho_Ladefoged_JPhon_2001_rfm.pdf
- Teeluk, Martha and Marie N. Blanchett. 1973. *Taiquallaruḡa = I Can Read*. Alaska Native Language Center, Fairbanks, Alaska.
- Tersis, Nicole and Michèle Therrien. 2000. Introduction. In Nicole Tersis and Michèle Therrien, editors, *Les langues eskaléoutes : Sibérie, Alaska, Canada, Groënland*, pages 15–29. CNRS, Paris, France.
- Uí Dhonnchadha, Elaine. 2003. Finite-state morphology and Irish. In *Proceedings of the Workshop on Finite State Methods in Natural Language Processing, 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 10), April 13–14 2003, Budapest, Hungary*, pages 43–49.
http://www.cl.cam.ac.uk/~ar283/eacl03/workshops03/W03-w9_eacl03dhonnchadha.local.pdf
- Webster, Donald H. and Wilfried Zibell. 1970. *Iñupiat Eskimo Dictionary*. Summer Institute of Linguistics, Fairbanks, Alaska. Digitized by Alaskool.org: <http://www.alaskool.org/language/dictionaries/inupiaq/default.htm>. The present work refers to the electronic version.
- Woodbury, Anthony C. 2002. The word in Cup'ik. In R. M. W. Dixon and Alexandra Y. Aikhenvald, editors, *Word: A cross-linguistic typology*, pages 79–99. Cambridge University Press, Cambridge, England and New York, New York.
- Yoo, Shinjae. 2008. A smart OCR for Inupiaq. Final presentation of graduate-level semester project in language and information technologies at Carnegie Mellon University.