University of Alberta

Morphological Parser for Field Linguists

by

Isabel Klint ©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Arts

in

Linguistics

Humanities Computing

Edmonton, Alberta
Spring 2007

Your file Votre référence
ISBN: 978-0-494-29885-5
Our file Notre référence
ISBN: 978-0-494-29885-5

Abstract

This thesis presents Parser X, a universal morphological parser for field linguists at the onset of a language documentation project. Parser X was designed, programmed and implemented after an evaluation of ten morphological parsers, four of which are based on Two-level Morphology systems. Two-level morphology parsers require grammatical information for implementation; therefore, they cannot be implemented at the onset of a project. An exemplar theory-based parser such as Parser X can be successfully implemented at the onset of a project. Two use-case scenarios describe the parser implementation in two transcription projects: the interlinearization of field data by a field linguist and the preparation of CHAT transcripts for CLAN analysis by a child language acquisition researcher using the CHILDES system. Both scenarios use Upper Necaxa Totonac language data. Parser X is a free and open-source parser, programmed in Java with data encoded in XML and suitable for field linguists.

Table of Contents

Table of Tables

Table of Figures

1 Introduction

A field linguist at the beginning of a language documentation project must collect and interlinearize texts in order to build up a corpus for his target language. Interlinearizing texts is a tedious task and much of it could easily be automated. Parsers are used by linguists to automate the breaking down of linguistic expressions (words, sentences) into constituent units. A parser is a tool that breaks language data into smaller elements, according to a set of rules that describe the data structure. A computational parser for natural language processing accepts natural language data and returns that data analyzed into elements according to rules; for example, a morphological parser decomposes words into morphemes. Parsers typically parse words by running a parsing routine that uses a lexicon and a set of morphological rules to identify the segments in the word and provide a morphological analysis of that word. Some parsers have a phonological processor that identifies segments that exhibit morphophonemic alternations.

Currently, tools that allow linguists to build their own parsers for well-studied languages are freely available or easy to obtain. However, despite the many advances in parsing techniques and the many parser-building tools available, only a handful of ready-made morphological parsers are available. Of these, only one or two are designed for field linguists in the process of documenting a language (for example, Shoebox

5, the Summer Institute of Linguistics' legacy software for field linguists). Currently there are no morphological parsers that are easy to find, download and implement for the major computer platforms. If the linguist were willing to implement a parser in his target language, he would discover that few parsers can parse all language types. The linguist in the early stages of a project may not know enough about his target language to decide if a particular parser can be implemented in that language or not. As endangered languages rapidly become moribund, a multi-platform and language independent tool to parse field data is needed. A parser for field linguists must meet the computational and linguistic needs of the field linguist.

Field linguists who wish to use a currently available parser will find that parsers are designed by computational linguists for a variety of different reasons. For example, parsers are designed by linguists to exemplify a theory, to parse a specific language, or for a specific purpose or type of user. Parsers that are theory specific have difficulty with linguistic phenomenon that are not accounted for in the theory; for example, the popular Two-Level Morphology model parsers have difficulty parsing words with segments that have long-distance dependencies. Furthermore, they require extensive rule sets and lexicons in order to be useful in practice. Language-specific parsers may be only applicable to one language or to languages that share a particular feature. For example, some parsers only parse suffixing languages and cannot parse prefixing languages. Parsers designed for a specific purpose or user may be difficult to implement for use in a different context without extensive programming knowledge.

Chapter 2 of the thesis introduces basic concepts in parser design. As many parser designs are described in academic papers (one or two are even available to download and implement), any new parser for field linguists must build from lessons learned. This thesis documents the design, implementation and usage of a parser for field linguists that can be implemented in any language. Chapter 3 gives an evaluation of ten parsers that inform the design of this universal parser. The ten parsers exemplify a variety of approaches to morphological parsing and a range of parser functionalities. The evaluation section describes how each parser meets needs specific to field linguists.

Chapter 4 describes the universal parser design and implementation. The parser's design is described in detail. The parser implementation in a polysynthetic agglutinative language is described. The parser's usage is described in two use case scenarios in Chapter 5. These scenarios describe the implementation and use of Parser X by student researchers. The first scenario describes a student field researcher's implementation of Parser X to interlinearize an Upper Necaxa Totonac text and the second scenario describes a student child language acquisition researcher's implementation of Parser X to correct and interlinearize child language data. Chapter 6 concludes this thesis with a thesis summary, a description of the universal parser's shortcomings and directions for future improvements to the parser.

2 Introduction to Concepts

2.1 *Criteria for Evaluation*

The first of the evaluation criteria is **parser universality**. A parser is universal if can adequately parse an agglutinative, polysynthetic language. The second of these criteria is the **parsing edge**, which refers to the starting point in a word from which the parser begins a parsing routine. By convention, this is the left edge, although some parsers parse from the right edge or from both edges. The third is **parser functionality** or the actions or processes that a parser performs. This might include a parser output such as an interlinearized gloss or an identification of lexical ambiguity. The fourth is **order dependent parsing.** A parsing routine that relies on the completion of a series of ordered steps is an order dependent parser. For example, in a case where the parser must parse a word with multiple affixes, if the parser identifies an affix that does not co-occur with certain other affixes, those other affixes are not included in the subsequent search for matches for the remaining affixes in the word. The fifth is **data incorporation**. For example, when a parser performs operations such as affix stripping on input words and then updates a lexicon to include novel roots whose class is determined by the stripped affixes, that parser is incorporating new data (new root and word class). A sixth criterion is **phonological processing**, where a parser decomposes input strings in accordance with phonological rules. The last is the **suitability for field**

linguists. The author of a particular parser may be aiming to illustrate a particular theory or implement a parsing algorithm that resolves a parsing issue present in a specific language; however, these aims do not always result in a parser that can parse raw field data. The author or programmer's goal in designing the parser affects the suitability of the parser for the field linguist. I describe each of these criteria in detail below.

2.1.1 Definition of Universal Parser

One goal of this thesis is to describe the suitability of specific parser designs for fieldwork. The design of a universal parser is constrained, as the name implies, by the requirement that the parser be able to parse any language. In order to adequately parse a typologically diverse set of languages, the parser must be able to parse languages that exhibit complex morphological or morphophonemic phenomena. Agglutinative, polysynthetic languages have complex morphology; therefore, these languages are typically avoided by computational linguists when building parsers as their complex morphology poses problems for the programmer. Languages that are morphologically complex include Turkish, Basque, Inuktitut and Nahuatl. For the purposes of this paper, a parser will be considered universal if it can adequately parse a morphologically complex language.

Although individual parsers differ in some respects, in general parsers incorporate the same basic features. These features form the criteria for evaluation of the parsers analyzed in Chapter Three.

2.1.2 Parsing Edge

The parsing edge refers to the beginning or end of the word to be parsed. In order to parse a word (represented by an input string), the parser must distinguish the root from the affixes. To identify the root, the parser must begin from one edge and iteratively check substrings of the input string until a root match is found. For languages that are suffixing only, such as Turkish, the problem of root recognition is reduced because the root edge is always the left edge of the word; there are no prefixes in Turkish. The root of the word is always word-initial; therefore, the left edge of the word is also the root-edge. Not only is the number of substrings that might be roots reduced but also the root substring location is known: only the root length and root match remain for the parser to discover. Consider example (1) from Hankamer (2006) below:

(1) **Turkish**

avrupalılaştırılamıyacaklardanşınızdır
avrupa–lı–laş–tır–ıl–a–mı–yacak–lar–dan–şınız–dır
Europe-ean-ize-CAUS-PASS-POT-NEG-FUT-ABL-2PL-POLITE-EMPH
'You are certainly among those who will not be Europeanizable'

In the above example, the text string that represents the word has 38 characters. The substring that corresponds to the root is the first six characters, *avrupa*. The parser must match a string of length *n* beginning from the left edge of the word to a root in the lexicon. The parser starts with the shortest substring (where $n = 1$) of *avrupalılaştırılamıyacak-lardanşınızdır* and continues until a root match is found. Table 1 shows the six substrings that the parser uses as search strings before finding a match in *avrupa.*

Table 2-1 Left-to-Right Parsing: Search Strings for Suffixal Language (Turkish)

NUMBER	SUBSTRING
1	a
2	av
3	avr
4	avru
5	avrup
6	avrupa

The algorithm searches for a word-initial root; if the algorithm reaches the end of the string before a match is found, then there is no root match.

Left-edge parsing is not always efficient for languages with prefixes. In the following example from Hoijer (1938, p. 3), this strategy is unsuccessful because the root is not word initial.[1]

[1] I provided the interlinear gloss; any errors are mine, not Hoijer's.

(2) **Chiricahua**

daanahibán
daa-nahi-bán
DISTR.PL-2DU.POSS-bread
'our/your bread'

In example (2), the root *-bán* is preceded by seven characters, the prefix string *daanahi-*. In this case, an exclusively left-edge parser fails to find a match in the lexicon for *-bán*. However, a parser using a left-to-right algorithm such as the brute force algorithm described in Charras and Lecroq (1997) would succeed in finding a match. Such a parser starts searching iteratively from the left-edge for a three-character root match, in this case for *bán,* beginning with the first three characters *daa* and continuing one character to the right with *aan* until it eventually matches the final three characters *bán*. A parser employing the brute-force algorithm would examine 51 substrings before finding the matching segment. This is illustrated in Table 2-2.

Table 2-2 Left-to-Right Parsing Search Strings for *daanahbán* (Chiricahua)

SUBSTRINGS OF *daanahibán*							
d							
da	a						
daa	aa	a					
daan	aan	an	n				
daana	aana	ana	na	a			
daanah	aanah	anah	nah	ah	h		
daanahi	aanahi	anahi	nahi	ahi	hi	i	
daanahib	aanahib	anahib	nahib	ahib	hib	ib	b
daanahibá	aanahibá	anahibá	nahibá	ahibá	hibá	ibá	bá
daanahibán	aanahibán	anahibán	nahibán	ahibán	hibán	ibán	bán

When parsing a language with prefixes, all prefix characters must be eliminated from the search before the root characters can be matched. In other words, the parser must find the root's location in the original word or input string. Therefore, it is more efficient to parse a language that only allows suffixes than it is to parse a language that allows prefixes.

In order to resolve the problem of root recognition for languages with prefixes, some parser designs include bi-directional parsing routines. In bi-directional parsing, the parser strips characters from the left edge and the right edge, either alternately or by parsing first from one edge then the other. For example, Neuval and Fulop's (2002) Whole-Word Morphologizer (WWM) will strip characters from the left or the right edge, depending on which edge shares characters with those of a word in the lexicon. Figure 1 shows a right-edge parse of the word *jumps* (Neuval & Fulop, 2002, p.6)

Input String:	jumps
Analysis:	xxxx+s
Output:	jump (V), jumps (3SG)

Figure 2-1. A Right-to-Left Parsing Example in English

In Figure 2-1, the WWM matches the last character of the input string *jumps* to the last character with the lexicon entry for *plays*, performs an analysis based on the parse of *plays* and outputs a parsed form for *jumps*. Neuval and Fulop's WWM analyzer implements a bi-directional matching function in order to perform a form of sequence-alignment, classifying input strings into groups based on the maximum number of characters the input strings have in common (2002, p. 4).

The practical reason for parsing bi-directionally, as described by Poibeau (1998, p. 110), is that bi-directional parsing improves parsing speed over uni-directional parsing for some strings. However, gains in parsing speed through bi-directional parsing differ from language to language. For a suffixal language, left-to-right parsing is more efficient. However, if a language has both prefixes and suffixes, a bi-directional parser may be more efficient than a uni-directional parser for some words. Specifically, a bi-directional parser is more efficient than a uni-directional parser in two instances. In the first instance, a bi-directional parser is more efficient when the left-edge segment has the length *n* and the right edge segment has the length *n*-1. For example, parsing from the right edge is more efficient for the input string *unhappy*, as shown below in Table 2-3:

Table 2-3 Right-to-Left and Left-to-Right Parsing: Prefixed Word and Efficiency

RIGHT-TO-LEFT	LEFT-TO-RIGHT		
y	u	n	h
py	un	nh	ha
ppy	unh	nha	hap
appy	unha	nhap	happ
happy	unhap...	nhapp...	happy

In Table 2-3, the right-to-left parse results in a match after five substrings while a left-to-right parse requires seventeen substrings to find a match. It is clear that a right-to-left parse would be more efficient in this case than a left-to-right parse. Bi-directional parsers, parsers that parse from both the left and right edge of the input string, can be more efficient than left-edge parsers when the parser recognizes the right-edge segment but not the left-edge segment. However, as is obvious from Table 2-3, a uni-directional parser will generate the same matches as a bi-directional parser. Notice that in the above examples all the parsing is performed by a variant of the Brute Force algorithm. Parsing efficiency will improve with other parsing algorithms such as the Boyer-Moore algorithm and its many variations. However, as parsing efficiency does not affect parsing success, it is not a criterion for evaluation and I do not further discuss the efficiency of parsing algorithms in this thesis.

Regardless of the implications of parsing edge on efficiency, the parsing edge remains an important criterion because it relates to the typological class of the language to be parsed. A universal parser must have a method for recognizing roots that does not rely on the parsing edge because without such a method no languages with prefixes could be parsed by that parser. Note that the parser may be theoretically bi-directional but in practice only parse from the left edge. For example, the Two-Level Morphology model-based parsers are in theory bi-directional parsers; however, they are implemented to parse from left-to-right. Therefore, these parsers, unlike a left edge parser such as Solak and Oflazer's parser (1993), are not dependent on the root edge coinciding with the parsing edge.

2.1.3 Parser Functionality

Parser functionality refers to the type of actions other than parsing that the parser performs. These include operations on input strings such as calculating the frequency of occurrence of each candidate parse per input string. For example, the Constraint Grammar Parser (Karlsson, 1995) incorporates frequency information in the parsing routine to return the most likely candidate for a parse. Parser functionality is a criterion for evaluation because parsers must perform some sort of analysis on input strings, for example to account for homographs (words that are spelt the same but differ in meaning) or to analyze phonological processes. Dolan's parser (1988) runs a pre-parsing routine that checks the input string for a sequence of characters that indicates a morphophonemic alternation. Without pre-parsing, it will be impossible to find the root as represented in the input string in the lexicon unless all variants of the root are listed in the lexicon. For example, the Indonesian prefix /məN/ (where *N* represents an unspecified nasal consonant) can cause a phonological change to the root word to which it is attached. Consider the following example from Dolan (1988, p. 81):[2]

(3) **Indonesian**
məminjam
mən-pinjam
TRANS-borrow
'to borrow from'

In (3), the root is represented by either the string *minjam* or the string *injam*. The parser must derive a prefix and a root in order to match these to the lexicon. For example, if the parser matches *mem* to a list of phonological variants of *meN* and *injam* to a list of phonological variants of *pinjam*, the pre-parsing routine could generate a string such as *menpinjam* for the underlying representation of the prefix and root strings. The revised string is entered into the parser to produce candidate parses. Typically, parser functionality is added to the parsing routine to account for some language-specific issue such as phonological alternations, lexical

[2] I provided the interlinear gloss, any error in it is mine not Dolan's.

ambiguity or floating diacritics such as in Classical Greek. However, in order to adequately parse all languages, a universal parser must also parse languages that have these phenomena. Parser functionality is therefore included as an evaluation criterion.

2.1.4 Order-Dependent Parsing

Order-dependent parsing refers to parsing that requires data to undergo operations in a specific order to maximize parser efficiency. A parser that recognizes a root's syntactic category (such as "noun") based on a successful prefix or root match will limit the set of possible matches for the unmatched affixes. For each successful affix match, other affixes are ruled out. For example, if an affix is a first person singular subject prefix, then no other subject prefix can attach to the root. Thus, the set of subject prefixes is removed from the set of legal affixes. Furthermore, if the subject prefix attaches to the verb, then all affixes that attach to nouns are also ruled out. The parser continues to search for matches from a decreasing set of legal affixes. This type of parsing is order-dependent because an affix or root must be matched before the set of subsequent legal matches are defined. In example 4, from Alegria, Artola, Sarasola & Urkia (1996, p. 4), each affix constrains the set from which the following affixes may be matched.

(4) **Basque**

daramazkiot
d-a-rama-zki-o-t
1SG-PRS-take.to-PL-DAT-3SG.IO
'I take things to him/her'

In the above example, the affix *d-* is first-person singular. The object cannot also be first-person singular; therefore, a legal string following *d-* can include only two possible subsequent affixes, the second- or third-person dative morphemes. If the string included a first-person dative morpheme instead, the string would be ungrammatical in this language. Assuming a left-to-right parse that begins matching segments to affixes or roots, the parse table for *daramazkiot* in Table 2-4 is plausible.

Table 2-4 Parse Table for *daramazkiot*

STRING	SUBSTRING	SEARCH DOMAIN	SUBSTRING MATCH	SEARCH DOMAIN MATCH
daramazkiot	d	all	d	1SG
aramazkiot	a	verb prefixes	a	PRS
ramazkiot	r	no match		
ramazkiot	ra	no match		
ramazkiot	ram	no match		
ramazkiot	rama	verb roots	rama	take.to
zkiot	z	no match		
zkiot	zk	no match		
zkiot	zki	verb roots	zki	PL
ot	o	verb suffixes	o	DAT
t	t	verb suffixes	t	3SG.IO

Bans are also possible. For example, the prefix *bait-* may be followed by morphemes belonging to a particular person paradigm, but those belonging to two other paradigms (not specified by the authors) are not permitted. Any restriction on the domain of possible segments improves the parser performance and eliminates the possibility of returning an ungrammatical candidate parse. As order-dependent parsing increases the accuracy of candidate parses, it is included as an evaluation criterion.

2.1.5 Data Incorporation

Data incorporation refers to the generation of new information during the parsing routine or the addition of new information to the parser databases (the lexicon, corpus or morpheme list). For instance, new information could be generated during the parsing routine when input strings are altered to reflect phonological processes. Data incorporation such as the addition of new roots to the lexicon is a form of artificial learning because the parser's output to a given input may vary over a number of texts as the output is determined by both the form and frequency of any string in the parser corpus. Data incorporation is useful because it allows the user to expand the lexicon, corpus or other parser databases. It is a criterion for evaluation because it is an important feature that gives the user control of the parser's knowledge base. Processes that require data incorporation are often order-dependent.

2.1.6 Phonological Processing

Phonological processing can occur either before or during the parsing routine. Some parsers have a separate routine for pre-processing input strings to rewrite input strings to account for any phonological processes. For example, *selves* may be rewritten as *self+s*. Other parsers perform phonological analysis as part of the parsing routine. Phonological processing may be considered a type of data incorporation in that the input string is re-written, generating a new string.

2.1.7 Suitability for Field Linguists

The objective of the parser designers is not necessarily user-driven. As shown by Anderson (1988, p. 5), the objective of parser design is generally to showcase a particular theory or to resolve a parsing issue in a specific language. Other reasons for building a parser are to create a computationally efficient parser or to build a parser tailored to the needs of a specific user. Therefore, not all parser designs are suitable for the field linguist. A parser for field linguists must meet certain computational needs. From a computational standpoint, the field linguist requires a parser that is platform independent (that is, that runs on the IBM, Apple and Unix platforms) so that the linguist is not limited to a particular platform. Furthermore, a platform independent parser would allow a wider community of users to share data or collaborate on a project. A parser for field linguists must be freely or at least easily available. Ideally, a parser for field linguists would be extensible or modifiable by the user without requiring programming knowledge to implement in order for the average field linguist to use the software. The parser should also be computationally efficient: it should not disrupt the computing environment by adversely affecting the processing power available to other applications.

Several programming tools are available for linguists with programming ability who want to build a computationally efficient parser. The Xerox tools form the basis of many parsers (for example, the ALEGRIA parser). The SOLAK parser was built with Unix tools Lex and Yacc. Other kinds of tools include MinorThird (Cohen, 2004), the Alembic Workbench and Ellogon (Petassis, 2005); however, only the first of these includes the tools to build a morphological parser. The major drawback of these tools is that they require programming knowledge to implement.

A parser for field linguists should also meet linguistic requirements. Primarily, such a parser must be language-independent. As field linguists often start a documentation project with little or no language data, a parser for field linguists should not require a large corpus, lexicon or grammar. As languages often require special characters, the parser must be able to accept and display these characters. For example, a Unicode encoding such as UTF-8 can display most characters. It is important that the field user be able to begin using the parser without having to learn a special encoding format or navigate a complicated interface. The parser must be designed with ease of use as a primary goal.

The original parsing objective of the designer can limit the suitability of a parser for field linguists. The type of parser a field linguist would require must meet certain computational and linguistic criteria. The parsers evaluated in Chapter Three are described in terms of their suitability for a field linguist at the beginning of a language documentation project who requires a parser for morphological analysis.

3 Evaluation of Parsers

The parsers analyzed in this thesis were chosen because the authors claim that these parsers adequately parse a typologically diverse set of languages. In other words, each parser can parse languages that have complex morphological or morphophonemic phenomena. This section contains the analysis for the following ten parsers (where the parser has no name, the name of the first author, in capitals, stands for the parser name):

ALEGRIA (Alegria, Artola, Sarasola & Urkia, 1996)[3]

BITC (Sprouse, 2000)

CGP (Karlsson, 1995)

DOLAN (Dolan, 1988)

KOVAL (Koval et al. 2000)

PC-KIMMO and K-Text (Antworth, 1993)

Morpheus (Crane, 1991)

Qpop (Wallace, 1988)

SOLAK (Solak & Oflazer, 1993)

Whole Word Morphologizer (Neuval & Fulop, 2002)

Each parser design is evaluated according to the criteria described in Chapter Two. If the parser is available for testing, a description of the

[3] ALEGRIA is officially known as Morfeus (also spelt Morpheus), the Basque parser described but not named in Alegria et al. (1996). Morfeus is here called ALEGRIA to avoid confusion with the Morpheus parser described by Crane in 1991.

parser implementation is included. If the parser is not available then I evaluate it based on the author's report. As several of the parsers evaluated in this section are based on the Two-Level Morphology model of language, this chapter is divided into two sections: the first section describes Two-Level Morphology and parsers based on a Two-Level Morphology model of a language and the second section describes the remaining parsers.

3.1 Two-Level Morphology Parsers

Many of the parsers I analyze in this chapter are based on the Two-Level Model of Morphology model of language (TWOL), first described by Johnson (1972) and Koskenniemi (1983, a and b).[4] In a TWOL model, "a word is represented as a direct, character-to-character correspondence between its underlying form and its surface form" (Antworth, 1993, p. 391). The two basic components of a TWOL model of a language are a lexicon of strings and a set of rules that maps inflected strings from their surface forms to their underlying forms. In this section, I briefly describe the TWOL model of morphology that forms the basis of the parsers analyzed below.

[4] A more detailed discussion of Two-Level Morphology models may be found in Chapter One of Jurafsky and Martin's *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2000).

3.1.1 Lexicon

The lexicon is a network of finite state automata (FSAs). An FSA is a regular expression or a "formula for expressing ... a class of strings" (Jurafsky & Martin, 2000, p. 22). For example, the FSA in Figure 3-1 represents the string *dog*:

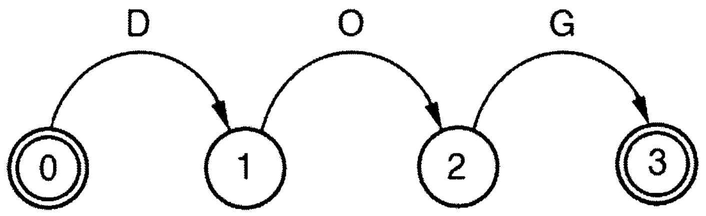


Figure 3-1. Finite State Automaton of *dog*.

The FSA in Figure 3-1 has four states: the starting state 0, two transition states and the final state 3. Each transition between states is represented by an arc between two states of the FSA. From the initial state 0, the FSA can only move to the next state 1 if the first character of the input string matches the legal path *d*. The second character *o* is also a legal transition as is the final character *g*:. Here, the FSA reaches the final state. If the parser reaches the final character but cannot make a legal transition, it cannot reach the final state. For example, if the input string were *dof* it would be an illegal string in the language because there is no path that leads from state 2 (*do*) to the final state 3 (*dof*). Several FSAs can be combined to create a finite state network (FSN). In a TWOL model of a language, each character in the input string is verified as a legal step through the network of FSAs, until the end of the string is reached (a variety of different programming techniques can perform these steps). If the input string cannot be matched to a legal path from the beginning and ending states on the network, it is an illegal string. In Figure 3-2 (adapted from Beesley & Karttunen, 2001), a path in the network is illustrated.[5]

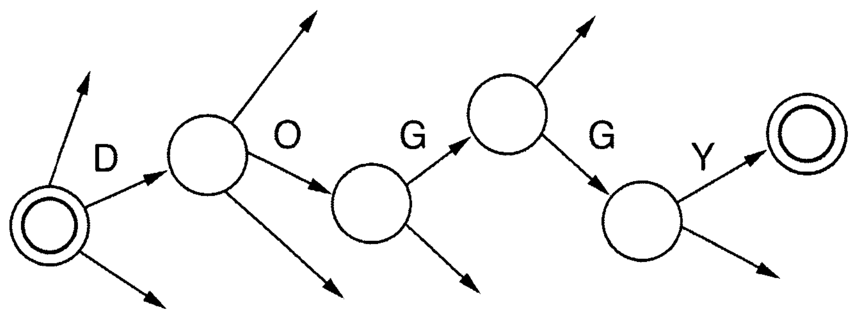


Figure 3-2. A Path in the Network

From the starting state, the string *doggy* is matched character by character against a possible path through the network. From each state, a number of possible paths extend. If the string being matched were *doggie*, and that word were not in the lexicon, the string would fail to find a path from *dog* to *gy*. The string would be identified as an illegal string. *Doggy* is a legal string because the final state is reached. The lexicon does not contain all legal strings in the language; typically, the lexicon only contains root strings and other strings that cannot be analyzed. TWOL systems are powerful because inflected strings (strings that contain affixes, for example) do not need to be encoded in the network but are recognized through the addition of the rules component described below that

[5] Figure 3-2 is adapted from Beesley & Kartunnen, 2001.

specifies which lexemes may be combined with affixes, thus adding new paths to the existing paths in the network.

To avoid encoding every possible string pair, each root is associated with a word category; for example, the root *dog* has the word category *noun*. Additionally, each root is associated with a continuation class. Koskenniemi (1983a) describes continuation classes as the morphotactics of a language. In a TWOL model, any morphosyntactic information included in a lexicon entry is an assignment to a continuation class. Continuation class information for each root is listed with the root in its entry in the lexicon. For example, entries for the roots 'talk' and 'walk' are shown in Table 3-1 and continuation classes for verbs and nouns are shown in Table 3-2.

Table 3-1 Lexicon Entries

ENTRY	ROOT	CLASS
1	Talk	Verb
2	Walk	Verb

Table 3-2 Continuation Classes

CLASS	CONTINUATION CLASS
Verb	-ed, -ing, -s, -#
Noun	-s, -#

If a root has the class 'Verb', it cannot have affixes of the class 'Noun'. As a parser moves from state to state in a network, the continuation class information associated with a lexeme indicates to the parser the legal path on which to continue.

3.1.2 Rules

In a TWOL model of a language, inflected strings do not appear in the lexicon. For example, the string *dogs* would not appear in a lexicon for English, yet it is a legal surface string in this language. The FSA that represents the underlying representation of dogs is shown in Figure 3-3.

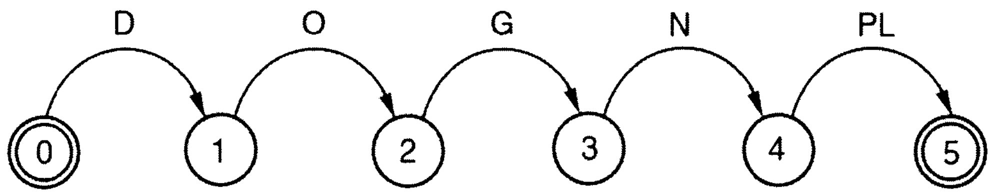


Figure 3-3. Finite State Automaton of dog.N-3p.

This FSA differs from the FSA shown in Figure 3-1 in that it shows the lexical or underlying representation of *dogs*, while Figure 3-1 shows the orthographic representation of *dog*. The correspondence between the orthographic representation and underlying representation of *dogs* is illustrated in Figure 3-4. Figure 3-4 shows the correspondence between the two representations of *dogs*.

Orthographic representation	d	o	g	ε	S
	↕	↕	↕	↕	↕
Underlying representation	d	o	g	N	PL

Figure 3-4. Correspondence Between Two Representations of *dogs*.

Every character of the orthographic representation string is mapped to another character on the underlying representation string, showing a one-to-one correspondence between the two strings. Note that ε in the surface string corresponds to *N* in the underlying string. The automaton that performs the mapping between two strings is another kind of finite automaton called a finite state transducer (FST). It can output a lexical or underlying representation when given its orthographic representation (and vice-versa). The string *dogs,* while not in the lexicon, can be verified as a legal string by an FST that maps paths for the inflected strings through the network. This FST is composed of two FSTs: one that matches roots to root-classes, verifying that *dog* is a regular noun and a second FST that verifies that *dog* can take the plural *s*. A simplified diagram of this FST is shown in Figure 3-5.

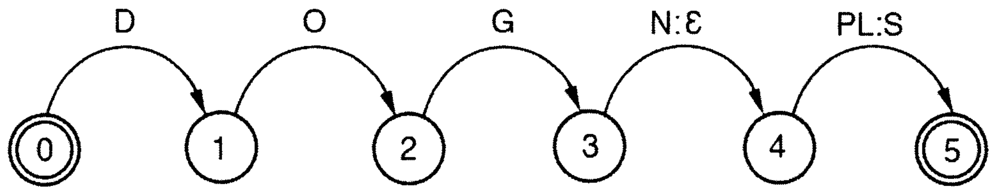


Figure 3-5. A Finite State Transducer for *dogs.*

Although the string *dogs* does not appear in the network, the FST in Figure 3-5 shows how the continuation class info allows the FST to select a legal path through the network for the inflected strings.

Each TWOL model is a network of all the legal strings in a language and series of rules. Any parser implementation that includes a TWOL model is language dependent; however, the basic parser design may be language independent if the parser is modular and the lexicon and rules are separate from the parser engine. For example, the Xerox tools can be downloaded and implemented in several languages for which the user supplies a lexicon and rules.[6] One of the disadvantages of the TWOL model is that if the lexicon and rule set must be supplied by the user in order for the parser to function correctly, the user cannot use a TWOL parser at the beginning of a data collection project to perform parses because the lexicon and rule set are incomplete. TWOL model parsers remain popular because they are language-independent. The TWOL parsers evaluated here are implemented in several languages with the exception of one parser that is an implementation of the Xerox tools.

3.2 Evaluation of Two-Level Morphology Parsers

3.2.1 PC-KIMMO and K-Text (Antworth, 1993)

Antworth's objective in designing PC-KIMMO is to create a freely available implementation of a TWOL model parser. PC-KIMMO can be downloaded and installed on a personal computer running the MS-DOS, Mac Classic or Unix operating systems. The finite state tables that create the rules for generating surface representations from underlying representations or vice-versa can be generated by K-Gen, another

[6] These tools are described in detail in Beesley and Kartunnen, 2001. The book includes the software. The software can be downloaded from the book's website:
<http://www.stanford.edu/~laurik/fsmbook/home.html>.

downloadable program. Also available is K-Text, a program that works in conjunction with PC-KIMMO. K-Text is a text editor for linguistic corpora. K-Text produces either parsed data for further analysis or interlinear glosses. Thee programs are still available for the PC from SIL's website. These programs are all freely available and open source. Furthermore, they are based on user-created language files.

The PC-KIMMO parser is not language-dependent; rather, the user is able to add the language files to the parser and update those files at any time. PC-KIMMO is unidirectional (it parses from the left-edge of the word). The parsing routines are as follows:

Table 3-3 PC-KIMMO Output

PC-KIMMO OUTPUT	
Unambiguous String	
Glossed string	V(hope)+PROG
UR	hope+ing
SR	hoping
Ambiguous String	
Glossed string	%2%N(spy)+PLURAL%V(spy)+3SG%
UR	%2%spy+s%spy+s%
SR	spies

As shown in Table 3-3, when the input is the unambiguous string *hoping,* PC-KIMMO parses the input and returns the interlinear gloss, *V(hope)+PROG*. For the input *spies,* an ambiguous string, PC-KIMMO returns two analyses. The first analysis is *N(Spy)+PLURAL* and the second analysis is *V(spy)+3SG*. PC-KIMMO parses concatenative morphology and K-Text decomposes forms that exhibit morphophonemic alternation. K-Text can pre-parse input to regularize spelling or transform orthographic transcription into phonetic transcription.

PC-KIMMO, together with KGEN and K-Text is easy to download and implement. However, it is difficult to use because it runs from the command line. The user must learn the commands and notation format in order to enter the rules that map SRs to URs. In addition, the user must run several programs in conjunction with PC-KIMMO. A basic set-up for the field linguist would be the following: PC-KIMMO, KGEN, K-TEXT, Text Editor and a video or audio player. Furthermore, in order to use the program, the field linguist would need to create several rules files. However, as a parsing tool PC-KIMMO is adequate. PC-KIMMO has the advantage the TWOL model affords, the computational efficiency of having a rule set that is separate from the lexicon. Furthermore, PC-KIMMO does not require a complete lexicon or set of rules to run.

3.2.2 CGP (Karlsson, 1995)

The Constraint Grammar Parser (CGP) is a TWOL model-based parser implemented in several natural languages and designed to embody a Constraint Grammar description of a specific language (Karlsson, 1995, p. 11). In 1994, there were CGP implementations for English, German, Swedish, Finnish, Danish, Russian and Estonian. CGP has three main functions: to pre-process parser input, to maintain surface ambiguities in parser output and to use both TWOL system parsing and probabilistic parsing (Karlsson calls this 'guesser' parsing). CGP is based on a TWOL model; therefore, it is theoretically bi-directional. However, the parser implementations all parse from the left edge of the word.

The parser accepts "pre-parsed" input: input that has been marked-up for word class, as well as other information. The parsing routine requires part-of-speech information; therefore, the input string is manipulated by a pre-parsing routine. In this way, the parser is order dependent because the input string is altered by a parser module before it is analyzed. The parser matches the pre-parsed input string against a master lexicon: the continuation classes for each lexeme, which contain all possible inflections or derivations for a lexeme, are listed with the lexeme. The parsing routine is an interleaving of a TWOL system parser and a "guesser". When the parser fails to parse based on the rules and the lexicon, the guesser assigns a parse to the words rejected by the TWOL system parser by applying ordered rules to create an output. The ordered rules can refer to both the form and the context of the rejected string. Multiple 'guesses ' or candidate output forms are returned to the user for acceptance.

While Karlsson describes a parsing philosophy that gives the user optional control over many functions of the parser (such as ambiguity resolution), the user must implement those functions in a language-specific way for his particular parser. Therefore, each parser implementation is language-dependent although the model and design are not.

Karlsson's main objective in designing a parser is to implement Constraint Grammar theory; therefore, the disambiguation of morphological strings is the parser's main task (p. 25). Karlsson claims that computational efficiency is gained by maintaining surface ambiguities in the parsing output. For example, the parse of a sentence such as 'Bill saw the little dog in the park' would assign two syntactic labels to 'in the park', maintaining ambiguity. Karlsson states that in maintaining surface ambiguity the parser processes less information and is therefore more efficient.

Furthermore, Karlsson claims that this manner of processing information is psychologically realistic, as is probabilistic parsing and the optimization of the lexicon to include low-frequency words, new words and proper names specific to a text being parsed. However, despite this claim of psychological plausibility, Karlsson is forced to make a design compromise based on computational efficiency. The lexicon must also list words that are morphologically ambiguous to exclude them from parsing analysis. See Karlsson's example parses for the Swedish word *frukosten* in (5), below.

(5)

i) frukost + en	'the breakfast'
ii) frukost_en	'breakfast juniper'
iii) fru_kost_en	'wife nutrition juniper'
iv) fru_kost + en	'the wife nutrition'
v) fru_ko_sten	'wife cow stone'

In example (5) (Karlsson, p. 20), the string *en* matches both an affix and the low-frequency noun 'juniper'. Without a method for constraining output, the CGP parser will analyze *frukosten* as a compound noun that contains the low-frequency noun *en* 'juniper'. Karlsson states,

> In order to avoid excessive over-generation of spurious compound readings [such as (5) iii, above)], the noun EN 'juniper' must not participate freely in the productive process of compound formation. Consequently, nouns containing the noun EN as non-first element must be listed in the lexicon. (p. 28)

By including all compounds of low-frequency words in the lexicon, the lexicon increases in size. This example illustrates that although in theory, the lexicon and rules are separate, in some instances, rules are inefficient and it is preferable to simply list forms in the lexicon. However, such design decisions are motivated, not by psychological reality, but by computational efficiency. In the case of the CGP parser, lists are not created in order to conform to any theory of storage in the mental lexicon but rather to ease the parsing process.

Further compromising his ideal of psychologically realistic parsing, Karlsson admits that in order to achieve computational efficiency, the more infrequent homonym of any homonymous pair should not be considered productive, as in the case of the noun 'en' above. Therefore, the more infrequent homonym will have all forms listed in the lexicon.

Karlsson claims that the parser is a universal parser. He states,

> The formalism should have no bias in favour of some particular language type, and it should, in a demonstrable way and without ad hoc changes to the formalism or the program code, be applicable to several languages belonging to different language families. (p. 3)

In fact, the CGP can parse languages of various types. Unfortunately, Karlsson's descriptions of the parsing routine do not include examples drawn from agglutinative, polysynthetic languages.

Another of Karlsson's design objectives is to design a parser that is easily available to the research community. He states that anyone who sends him a 300-word text will receive the parsing results for that text (p. 18). However, this makes the results of the parser available, not the parser itself.

3.2.3 KOVAL (Koval et al. 2000)

The KOVAL parser is a parser for English, Portuguese, Russian, Turkish, Japanese, Finnish and Arabic, designed primarily to allow morphological analysis on an IBM-compatible desktop computer. The KOVAL parser is in principle a TWOL system (Koval et al., 2000, p. 145), although the finite state network is created at runtime, not encoded state by state into a network. For example, in a traditional TWOL system, a parser would move, character-by-character from state to state, whereas in

KOVAL, each transition in the network is made by a legal string or substring (root or affix) and each state is a step in the parsing process.

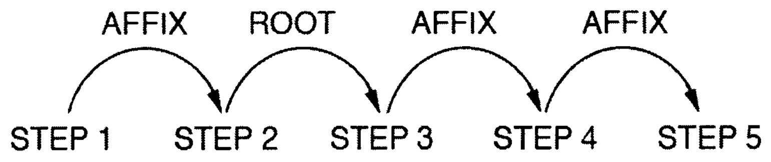


Figure 3-6. KOVAL as a Virtual Finite State Automaton

Koval et al. describe how the KOVAL parser creates a virtual finite state network at runtime:

> The notion of continuation classes is widely exploited in [Linguistic Automaton] morphotactics and the tabular representation of ranked affixes is nothing but a source for compiling a virtual finite state automaton [at runtime]. (p. 145)

The authors remark that their method of morphological analysis deviates from the TWOL model in some respects due to processing constraints. The authors state that these deviations from the model will be resolved in future versions of KOVAL.

In the parsing routine, KOVAL relies on the assumption that the left-edge is the root edge. Once the root is identified, each segment can occupy an ordered 'slot' in relation to the root. A slot may remain empty. Only one affix may occupy a slot (each set of legal slot occupants is a morphological paradigm). The content of these slots are matched against the suffix tables at runtime. In this way, the KOVAL parser is order-dependent.

Koval et al. make no sweeping claims that the KOVAL parser is a universal parser. The authors state,

> The experience of the... group's work on the upgrading of the first-generation [machine translation] systems revealed some general principals of creating computer morphology common to most languages. (Koval et al., 2000, p. 133)

Koval et al. go on to state that their system is best suited to languages with suffixal morphology.

3.2.4 ALEGRIA (Alegria et al. 1996)

Alegria et al. (1996) describe the automatic morphological analyzer that is the core module of tools such as XUXEN, a ing-checker for Basque described in Aduriz et al. (1997) and Agirre et al. (1992). The spelling-checker requires a morphological parser in order to correctly identify legal root and morpheme combinations.

ALEGRIA is 'designed with the objectives of being neutral in relation to linguistic formalisms, flexible, open and easy to use' (Alegria et al., 1996, p. 198). Ease-of-use is an important design goal because it places the needs of the user above other considerations such as computational efficiency.

ALEGRIA was built using the Xerox tools (for more information see Beesley and Karttunen's Finite State Morphology, 2003). The parser is a TWOL-based parser for Basque designed to parse a variety of Basque dialects, and it extends the TWOL system in order to account for long-distance morphological dependencies (such as circumfixation). In order to account for non-contiguous segment dependencies, Alegria et al. extend the TWOL model's continuation classes by adding bans and continuation trees (Alegria et al., p. 9). The bans are restrictions on the combinations of morphemes. For example, a continuation class lists the continuation classes with which it does not co-occur. The continuation trees are rules that specify alternative paths through a network for words that include a given morpheme. This is to propagate the rule throughout the network without encoding it repeatedly. Continuation trees are an example of order dependency. When a root grammatical category is recognized, the continuation class associated with that category restricts the search domain for continued morphological parsing.

Unlike other TWOL parsers, ALEGRIA relies on a lexical database (described in Agirre et al., 1995) instead of a lexical network. ALEGRIA allows user input into the parsing process. The most important difference between ALEGRIA and other TWOL parsers is that the user can update the lexicon. The user has access to a 'user lexicon' that can be updated by the user or automatically updated by the parser. New words are added to either the user-specific lexicon or the general lexicon. Allowing user input gives the user control of the lexicon.

Complex rules are used to manipulate input strings or decompose phonological processes such as morphophonemic alternations and orthographic processes. These manipulations are evidence of order dependency.

ALEGRIA is able to parse dialectal variations of words in the general and user lexicons by implementing a set of "non-standard" morphemes and a set of rules that account for regular morphological variations and common competence errors. For linguists working in a language with dialectal variation, this flexibility is highly useful.

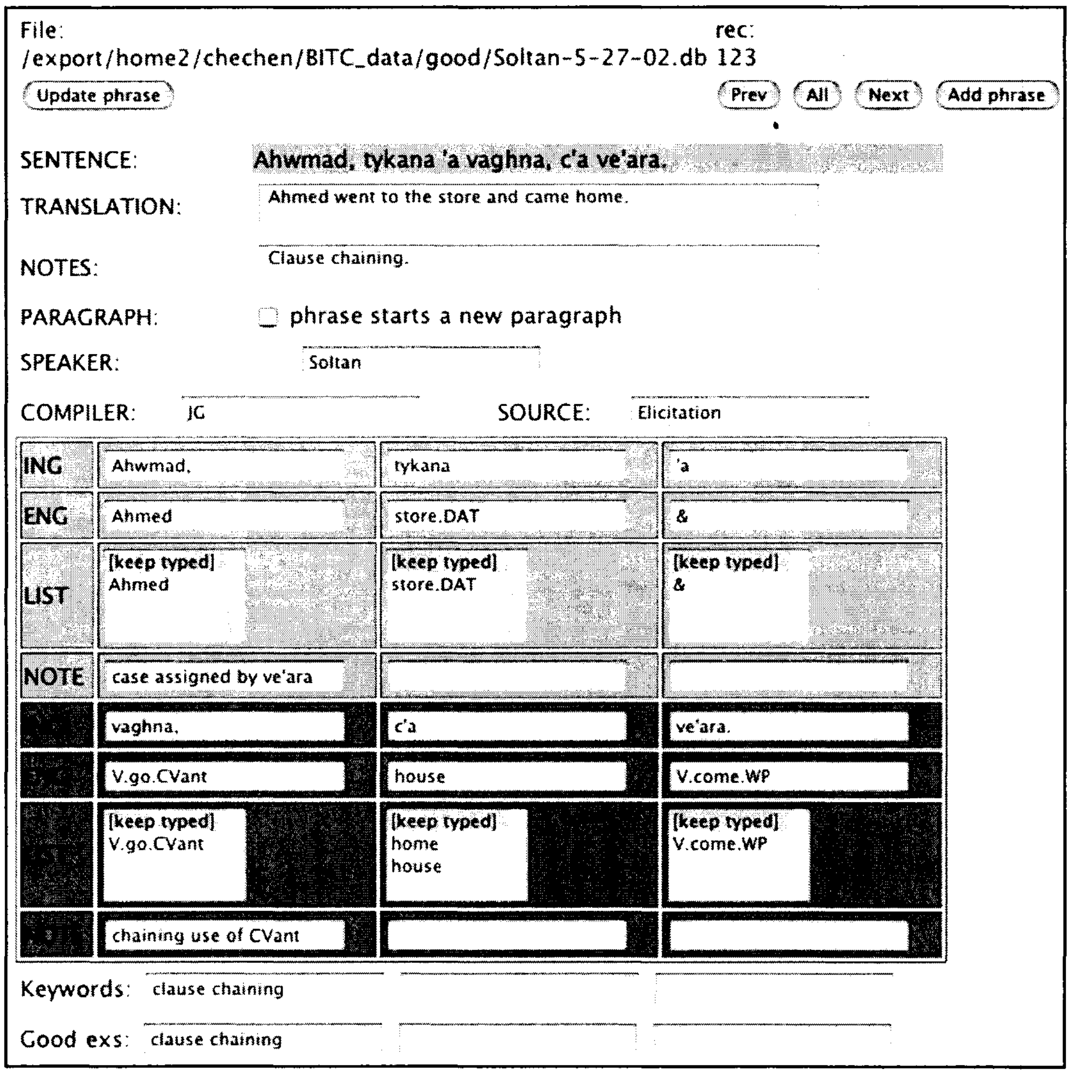
3.3 Other Parsers

The parsers in this section, unlike the parsers in the previous section, do not share a theoretical model. These parsers were designed either for a specific purpose (a class resource), to exemplify a particular theory, or to parse a specific language. A varied group of parsers will better inform the design of a new parser than a set of similar parsers.

3.3.1 BITC (Sprouse, 2000)

The objective in creating BITC was to provide a resource for multiple field linguists working on collecting and interlinearizing texts for an endangered language with complex morphology. Users can create new files such as concordance and search result files using BITC.

Although BITC is included here for analysis, BITC is a not a true parser in that it does not decompose or analyze forms. BITC relies on the user to provide functionality normally provided by other parsers such as root recognition, discovery of novel forms and decomposition into segments. Although BITC's functionality is limited to matching the input forms against a corpus and outputting interlinearized forms, BITC provides the user with a simple and useful interface that returns matching forms from the corpus or the lexicon. The BITC GUI is shown in Figure 3.7.[7]



File: /export/home2/chechen/BITC_data/good/Soltan-5-27-02.db rec: 123

SENTENCE: Ahwmad, tykana 'a vaghna, c'a ve'ara.

TRANSLATION: Ahmed went to the store and came home.

NOTES: Clause chaining.

PARAGRAPH: ☐ phrase starts a new paragraph

SPEAKER: Soltan

COMPILER: JG SOURCE: Elicitation

ING	Ahwmad,	tykana	'a
ENG	Ahmed	store.DAT	&
LIST	[keep typed] Ahmed	[keep typed] store.DAT	[keep typed] &
NOTE	case assigned by ve'ara		
	vaghna,	c'a	ve'ara.
	V.go.CVant	house	V.come.WP
	[keep typed] V.go.CVant	[keep typed] home house	[keep typed] V.come.WP
	chaining use of CVant		

Keywords: clause chaining

Good exs: clause chaining

Figure 3-7. BITC Graphical User Interface.

The GUI indicates that the user is interlinearizing a sentence in Ingush. Notice that the second word in the sentence, *tykana*, is being interlinearized as 'store.DAT' and that the list of results for *tykana* in the corpus includes 'home' and 'house'. Each successful match from the

[7] Figure 3-7 is reproduced from Good & Sprouse (2003).

corpus includes the corpus forms as well as other data such as its metadata. An example is the transcriber (called the compiler in the GUI above). In this way, BITC matches all words identified by the user to a corpus of words. The user can choose from previous interlinearizations for the input form. Any novel forms discovered by the user are added to the lexicon and the parsed and interlinearized text is added to the corpus.

The BITC module can be installed on a personal computer for individual research or on a network server for group collaboration over the Internet. Consider the example record from Good & Sprouse (2000) reproduced in (6):

(6)

```
<record id=1>                        # usually corresponds to a sentence
<source>Pacchahw                     # source of this record
Liir</source>
<compiler>JN 11-22-                  # identifies who compiled the record
99</compiler>
<speaker>Liir</speaker>              # identifies person speaking
<incl_level>1</incl_level>           # records compiler's confidence in the
                                     analysis
<translation>                        # free translation of whole record
   I can't speak.</translation>
<word id=0>Q'ameal</word             # first word
id=0>
<interlin id=0>                      # aligned interlinear gloss of first word
   conversation</interlin id=0>
<word id=1>dulac</word id=1>         # second word
<interlin id=1>                      # aligned interlinear gloss of second
   D.do.POT.NEG</interlin            word
id=1>
<note id=1>                          # note concerning second word
   /dielac/ is how this form is
actually
   pronounced.</note id=1>
</record id=1>
```

This example (enclosed in a <record /> tag) is a line from the play *King Lear* (*Pacchahw Liir* in Ingush). The researcher who collected this data (<compiler />) and the date of collection are recorded. The record includes the information that the speaker is Lear (this refers to the speaker in the play, not the language consultant providing the transcription) and the confidence level (<incl_level />) of the compiler for this record is 1 (highest confidence in transcription). The record shows the line of text, (<word />), the interlinearization (<interlin />) and the gloss (<translation />). A note regarding the line is also included in the <note /> tag. This example record clearly shows BITC's underlying eXtensible Markup Language (XML)-like tagging scheme. Using eXtensible Stylesheet Language Transformation (XSLT) transformations, users can transform a database produced from texts entered through BITC into an XML database. Furthermore, BITC is open-source software under the Artistic License and the source code is freely available from the author via the Ingush Project website. With some PERL programming ability, researchers can implement BITC for any language. The separation of the text and lexical databases from the collection, formatting and search modules of BITC allows researchers to benefit from corpus-based interlinearization tools while maintaining separate text and lexical databases.

3.3.2 DOLAN (Dolan, 1988)

DOLAN is designed to parse Indonesian. Indonesian has a small number of phonological and morphophonological rules and a small number of affixes. These affixes are monosyllabic (Dolan, 1988, p. 89). The DOLAN parser runs a pre-parsing routine to rewrite the input string to 'undo' any phonological processes. The parser then performs a syllable-based parsing routine that uses information about long distance dependencies between morphemes to produce a candidate parse.

The parser moves from left to right across the word, categorizing each syllable as a potential affix or root (or part of a root). All possible combinations are candidates. The parser then performs a second routine that searches for the root. If the root is found, a third routine assesses the grammaticality of the remaining candidates. The parser output is all candidate parses that have legal affix and root combinations.

For each input string, DOLAN performs a 'pre-parse' routine that rewrites the input string. Any input string must be syllabified by the user and then input to the parser. The parser then re-writes syllables in groups of three; in this way the parser analyzes syllables in the context of their adjacent syllables. In Dolan's example (reproduced here as (7)), three candidates are generated for a two-syllable word. In (7a), the parser assumes that the surface form is the same as the underlying form. In (b) and (c), the parser applies phonological rules to generate candidate second syllables from the surface representation. [8]

(7)

(a)	mənu	
	mənu	
	ROOT	
(b)	mənu	
	məN	-nu
	PREFIX	-ROOT
(c)	mənu	
	məN	-tu
	PREFIX	-ROOT

A second pre-parsing routine checks for reduplication. At this stage in the parsing process, the parser has a list of possible root-affix combinations

[8] Example (7a-c) adapted from Dolan, p. 96.

generated from the application of phonological rules and the presence or absence of reduplication.

The parser now searches for the root. If a root is found, the syntactic class of the root determines the legality of subsequent affix matches. This step eliminates the large number of possible parses generated by the pre-parser that do not contain the matched root. Dolan provides an example syllabified input string, *bərkəliaran*. The pre-parse routine identifies six possible candidates, shown in (8).[9]

(8) **bərkəliaran**

(a)	bər	-kə	-liar	-an
	PREFIX	PREFIX	ROOT	SUFFIX
(b)	bər	-kə	-liaran	
	PREFIX	PREFIX	ROOT	
(c)	bər	-kəliar	-an	
	PREFIX	ROOT	SUFFIX	
(d)	bər	-kəliaran		
	PREFIX	ROOT		
(e)	bərkəliar	-an		
	ROOT	SUFFIX		
(f)	bərkəliaran			
	ROOT			

The parser then performs a root search for each candidate in (8). Of the six candidates, only one contains a successful root match; (8a) contains the root *liar*, 'free, wild'. The other candidates are rejected. The final parsing routine checks the remaining parse in (8a) against twenty-seven legal

[9] Example (8a-f) adapted from Dolan (p. 96).

morphological combinations. The morphemes in (a) can be grouped in the following ways:[10]

(**9**)

(a)	bər	-kə	-liar	-an	
	CIRCUMFIX	ADJ	ADJ	CIRCUMFIX	
(b)	bər	-kə	-liar	-an	
	PREFIX	PREFIX	ADJ	ADJ	
(c)	bər	-kə		-liar	-an
	PREFIX	CIRCUMFIX		ADJ	CIRCUMFIX

The parser checks the legal combinations for each candidate in (9), and checks for the possible groups generated by the forms (b-f) in (8). For example, the groups generated by (9a), PREFIX PREFIX ROOT SUFFIX, match the analyses in (10).[11]

(**10**)

(a)	bərkəliaran		
	bər	-kə	-liaran
	PREFIX	PREFIX	NOUN
(b)	bərkəliaran		
	bər	-kəliaran	
	PREFIX	NOUN	

In (10a), the noun is derived from a rule that declares that nouns are generated by the combination ADJ+ *an* and in (b), the noun is derived from a similar rule that holds that nouns are generated by the combination of adjectives with the circumfix *ke*+ROOT+*an*. However, (10a) fails a second application of the rules because *ke* cannot combine with

[10] Example (9a-c) adapted from Dolan (p. 96)
[11] Example (10a-b) adapted from Dolan (p. 98).

nouns. The successful parse is (10b), a legal parse, because by another rule *bər* combines with nominal stems to create intransitive verbs. As the parser runs a series of parsing routines, each one generating the input for the next, the parser employs order-dependent parsing.

Dolan states that his parsing approach has several advantages of over TWOL approaches. These advantages include the ability to parse words with long-distance dependencies; for example, DOLAN adequately parses circumfixes, infixes and reduplication by applying rules that constrain morpheme combinations. Dolan quotes Koskenniemi, stating that Koskenniemi admits, "that his system would require 'extensions or revisions... for an adequate descriptions (sic) of languages possessing extensive infixation or reduplication'" (Dolan, p. 89). Dolan also notes that his approach allows the parser to have access to the entire string during the parsing process, whereas the TWOL model only parses character-by-character or segment-by-segment.

Dolan's objective is to create a parser that is 'psycholinguistically plausible' and computationally efficient. However, the Dolan parser relies on language-specific features such as monosyllabic affixes and a small number of productive affixes. A language that does not share these features would be difficult, if not impossible, to parse with a similarly designed parser because morpheme boundaries do not coincide with syllable boundaries in every language. Furthermore, the necessity of encoding a rule-set for phonological processes and morphological combinations in order to perform parses rules out DOLAN as a model parser for field linguists.

3.3.3 SOLAK (Solak & Oflazer, 1993)

Solak and Olazer's parser, here called SOLAK, parses Turkish. The design objective is to create a morphological parser embedded within a spelling-checker for Turkish. The authors state,

> In order to check the spelling of a Turkish word, it is necessary to make significant phonological and morphological analyses. (Solak & Oflazer, 1993, p. 114).

The SOLAK parser can be embedded in other applications.

The parsing routine relies on the fact that Turkish is an exclusively suffixal language. In Turkish, the root always begins at the left-edge of the input string; therefore, the problem of root-recognition is trivial. The first step in the parsing routine is to identify the root. If no root is identified, characters are stripped from the right edge until a match is found in the lexicon. A vowel harmony check is then performed. The input string is then analyzed by either a noun or a verb parser (depending on the class of the root). If the resulting input string (root + affixes) does not result in a successful parse, a new root is sought by removing characters from the right edge of the previous root match until a new root match is made. Some string sequences indicate possible morphophonemic alternations; in these cases, strings containing these sequences are manipulated according to the appropriate rules during the parsing routine. In order for the manipulated input string to result in a successful root match, the root's entry must indicate that the root can undergo phonological processes in conjunction with the affixes indicated by the parsing routine.

The parser was built using pre-existing software tools (middleware). The authors incorporate the publicly available Lex and Yacc UNIX tools into the parser to perform part of the parsing task. Lex splits a source file into tokens and Yacc, given a context-free grammar, generates a parser. In SOLAK, Lex separates the input string into a root and a list of suffixes and Yacc parses the suffixes generated by Lex using a set of rules for Turkish provided by the authors (Solak & Oflazer, 1993, pp. 119-120). Two Lex and Yacc specifications are implemented in SOLAK, one for nouns and one for verbs. Lex and Yacc are now open-platform and open-source, and are available for download from Sun Microsystems as Javacc 3.2 (Sun Microsystems, 2003)). The vowel harmony checks and morphophonemic checks mentioned above are performed by other parsing routines.

SOLAK is designed to be extensible, although programming expertise would be required to modify the source code. The authors intended the parser to be embedded in other applications or used as a stand-alone application; they designed a text-editor for the UNIX platform that utilized the parser as part of a spelling-checker. However, as the SOLAK parser does not recognize new words or allow new paradigms or new roots to be added to the parser database, its usefulness for the field linguist is limited.

3.3.4 WWM (Neuval & Fulop, 2002)

The Word Formation Strategy/Whole Word Morphology parser for English and French (WWM) is designed to discover the morphological 'relatedness' of words. The design objective is to model human cognition based on the Whole Word Morphology or Seamless Morphology theory developed by Alan Ford and Rajendra Singh (Neuval & Fulop, 2002).

WWM relies on orthography to perform its analysis. Every word in the WWM's small lexicon of 1000 to 5000 words is listed with its syntactic and morphological category. Each word is then segmented into substrings based on orthographic similarities in substrings of other words. For example, the string *reception* is segmented into *rece* and *ption* based on the string *receive,* which is segmented into *rece* and *ive.* In parsing a string, WWM classifies substrings into sets; for example, the word-initial substring *rece-* may be classified into the set of all strings that are followed by the affix *-ive;* or the word-final substring *-ption* may be classified into the set of segments that are preceded by the string *rece-*. The authors describe this segmentation as an analysis that "parses any complex word into a variable and a non-variable component" (Neuval & Fulop, 2002).

The parser then acquires Word-Formation Strategies (a central term in Whole Word Morphology) from the relationship between these lists and a corpus of successful previous parses. As the parser discovers the differences between two words that share a segment, it learns morphological strategies for generating similar words. Therefore, if it has acquired *receive* and *reception,* it can recognize the relationship between *deceive* and *deception*. However, the Neuval admits that this strategy is limited for languages with complex morphology such as infixation or suppletion (Neuval, 2002, p. 459) or for discovering the relations between words such as *am* and *was* (Neuval & Fulop, 2002, p. 9).

The parser then generates new words based on the entries and word formation strategies it has discovered. For example, if the lexicon contains *perceive, receive* and *reception,* the parser will be able to generate *perception.*

The parsing edge, regardless of parsing technique, is bi-directional to allow partial matches of the whole form beginning from either edge. The parser's main functions are to derive word formation strategies and generate new lexicon items.

The authors distinguish their parser from TWOL systems on two points. Firstly, the parser does not decompose strings into substrings but rather classifies substrings into two categories. Secondly, the parser acquires word formation rules from the corpus and therefore a parse of an input string can change over time given new words in the corpus. However, the theory specificity of the parser results in a parser that cannot perform universally without modification.

3.3.5 Qpop (Wallace, 1988)

Qpop parses Bolivian Quechua, a suffixing language; therefore, root recognition is easily performed by searching for *n*-length substrings of the input string in the lexicon.

Qpop manipulates the input string to account for the application of any phonological rules. Matching the manipulated string against a lexicon produces candidate stems. The candidate stems are matched against a list of derivational morphemes. All successful matches are rewritten to 'undo' any phonological changes and then, if there are unmatched characters remaining in the string, the string is matched against a list of inflectional morphemes. All successful parses comprise the parser output. Thus, Qpop is order-dependent. Qpop's distinction between derivational and inflectional morphology is inherent in the parsing routine.

Wallace states that her parser design is based on Hankamer's approach to parsing based on morpheme ordering (1986). Due to the type of morpheme combinations in Bolivian Quechua, the parsing routines must parse non-contiguous segments. Wallace states (as did Dolan) that TWOL models are unable to handle these long-distance dependencies.[12] Wallace's parsing objective and parser design are not suitable for the field linguist because the language-specific parsing routines would be difficult to implement for other languages.

[12] For an up-to-date description of TWOL models and non-contiguous segment parsing strategies, see Beesley and Kartunnen, 2001.

3.3.6 Morpheus (Crane, 1991)

Crane programmed Morpheus specifically to perform powerful searches of Classical Greek texts. Crane states that using information retrieval methods to search Classical Greek texts required the addition of 'morphological intelligent retrieval tools' (Crane, 1991, p. 243). Morpheus uses a three-tiered function to perform morphological analysis of input strings to compensate for floating diacritics, suppletion and dialectal variation. Morpheus has since been implemented for Latin and Italian. However, the current programmers predict problems adapting Morpheus to languages that are not primarily suffixal. Indeed, plans to adapt Morpheus to Arabic, which has template morphology, include a new morphological parser engine (Mahoney, Rydberg-Cox, Smith & Wulfman, 2000).

Morpheus has three parsing strategies. Crane describes the first strategy, 'big bang', as a modified form of the Unix **fgrep** utility, a fixed-string search function.[13] 'Big bang' generates all possible word forms based on the root form of the input string and then uses FSAs to locate these in a corpus. The output is all matched word forms. The second strategy, based on David Packard's MORPH algorithm (Packard, 1977), is to strip affixes from the right edge of the word until a root can be matched against lexicon strings. The input string is manipulated and new strings are generated from which to continue parsing. The third strategy

[13] Fgrep is now obsolete. It is replaced in Unix by **grep -f**.

is a rule-based system that allows the programmer to add rules that apply to the system as a whole. Crane describes a hierarchy of 'generators' that can generate the majority of legal strings in the database, given the appropriate input strings. Crane envisions a user who inputs a string and receives as output all forms with that root in the database.

One drawback in the design is that it is difficult to add new forms. Crane states, "New words constantly exhibited peculiarities which forced us to revise our model of Greek morphology." These revisions required reprogramming of basic data structures (Crane, 1991, p. 245). This is a serious drawback because the user must be certain that all words in the language are included in the lexicon in order to implement the parser.

Morpheus has grown from a one-programmer, language-specific application in C++ to a multi-programmer Java application. Morpheus is integrated with the Perseus Digital Library (Crane, n.d.), the Suda On Line (Whitehead, 2001) and the Chicago Homer (Kahane & Mueller, n.d.). It is not expected that the Morpheus parser engine will be available for linguistic fieldwork applications. Nevertheless, the design philosophy, parser model and extensibility make Morpheus a good example of parser design.

3.4 *Results of Evaluation*

An analysis of the ten parsers described above provides insight on features a parser can provide the user, types of morphological analysis that a computational parser can perform, different parsing algorithms and different approaches to parser design.

3.4.1 Parser Universality

All parsers but one of those described above parse agglutinative, polysynthetic languages. Some of the parsers were implemented in languages from more than one typological class: these are universal parsers. Table 3-4 shows the languages parsed by the parsers evaluated above.

Table 3-4 Parsers by Languages Parsed

PARSER	LANGUAGES
ALEGRIA	Basque
BITC	Ingush
DOLAN	Indonesian
CGP	English, German, Swedish, Finnish, Danish, Russian, Estonian.
KOVAL	English, Portuguese, Russian, Turkish, Japanese, Finnish, Arabic
Morpheus	Classical Greek, Latin, Italian
PC-KIMMO	English, Finnish, Greek, Japanese, Hebrew, Kasem, Tagalog, Turkish
Qpop	Bolivian Quechua
SOLAK	Turkish
WWM	English, French

The parsers above are implemented for agglutinative, polysynthetic languages, with the exception of Morpheus, which is implemented for a fusional-synthetic language, Greek and WWM, which is implemented in English and French. PC-KIMMO, KOVAL, CGP, Morpheus and PC-

KIMMO are implemented in several languages. Four of the ten parsers are designed to parse a specific language; these are useful as models for specific design features rather than for specific parsing algorithms. The authors claim, for five of the parsers, that their parser can be implemented for almost any language. These are BITC, CGP, KOVAL, PC-KIMMO and WWM. Of these, CGP and KOVAL require large lexicons and grammars in order to implement and PC-KIMMO requires specialized knowledge of TWOL rule writing. Clearly, as all three have several implementations in a wide variety of languages, these parsers are of use in the linguistic community.

3.4.2 Parsing Edge

Despite the expectation that TWOL system-based parsers would take advantage of the bi-directional parsing theoretically available in the model, none of the TWOL parsers parse from right-to-left. Only two parsers parse from both the left and the right edge, the WWM parser and Morpheus. Seven of the ten parsers are dependent on the root edge being the left edge of the word (word-initial). Nine of the parsers perform an automatic analysis in search of the root; BITC is the only parser that does not search for the root but rather relies on the user to isolate the root in the input string. However, BITC does aid the linguist in performing interlinearizations. The graphical user interface (GUI) allows the user to input a word and segment that word using the lexicon and an affix table as resources. As each segment is typed into a table cell, BITC lists previous interlinearizations for that segment below it. This allows the user to select the appropriate interlinearization while maintaining a consistent interlinearization across the corpus for that segment. Although BITC does not perform an analysis and therefore does not have a parsing edge, as a parsing aid it searches from the left edge of each identified segment. The parsing edge of each parser is shown below in Table 3-5.

Table 3-5 Parser Edge of Each Parser

PARSER		ROOT EDGE	PARSING DIRECTION
TWOL Parsers	ALEGRIA	Unknown	→
	CGP	Unknown	→
	KOVAL	Unknown	→
	Morpheus	Unknown	↔
	PC-KIMMO	Unknown	→
Other Parsers	BITC	Unknown	No analysis
	DOLAN	Unknown	→
	Q-pop	Left-edge	→
	SOLAK	Left-edge	→
	WWM	Unknown	↔

The parsers that rely on the root edge to be the left-edge are not language independent, while parsers that do not rely on a left-edge root to successfully parse an input string are language independent. While the TWOL parsers are less likely to be language dependent (three of the five parsers are implemented in more than three languages), or to rely on word-initial roots (like Q-pop and SOLAK) or well-defined root edges (like DOLAN and BITC), TWOL parsers require that the user have a lexicon and rule set prior to implementation. After eliminating TWOL parsers and parsers that rely on word-initial roots, the remaining parsers are WWM, BITC and DOLAN. BITC requires user input to perform root identification. DOLAN's automatic parsing tests all possible candidate root forms.

3.4.3 Parser Functionality

The parsers have different functions that improve the accuracy of their parses (order-dependent and data-incorporating parsing and phonological processing are described below). DOLAN uses syllabification rules to better identify morpheme and root edges. This function is unfortunately language specific. The CGP parser uses frequency data to predict the best candidate parse. This is an excellent strategy, especially when coupled with other strategies to eliminate error when a seldom-used form is parsed. ALEGRIA and Morpheus can parse inputs with dialectal variation. ALEGRIA even allows the user to build dialect-specific lexicons. Allowing the user to parse different dialects of the same language means that the user does not require a separate parser for each dialect.

3.4.4 Order Dependency and Data Incorporation

The data-incorporating parsers are those parsers that can incorporate data from a parsing routine into their databases. For example, WWM acquires new word formation rules that are added to its databases. Of the ten parsers, nine have order dependent parsing and four have data-incorporation (where the author does not state that the parser allows the addition of new lexemes or rules to its database via the parser interface, I assume that it does not). These results are summarized in Table 3-6, below.

Table 3-6 Parsers by Data Incorporation and Order Dependency

PARSER	DATA INCORPORATING	ORDER DEPENDENT	TOTAL PARSERS
ALEGRIA	1	1	1
BITC	1	0	1
CGP	0	1	1
DOLAN	0	1	1
KOVAL	?	1	1
Morpheus	0	1	1
PC-KIMMO	1	1	1
Qpop	?	1	1
SOLAK	0	1	1
WWM	1	1	1
TOTAL PARSERS	4	9	10

The inability of a parser to acquire new forms in any other way than by manually updating the lexicon, rule set or other parser database is a serious limitation. Only ALEGRIA allows the user to easily add new lexemes to the database via the parser.

3.4.5 Phonological Processing

Of the parsers evaluated above, seven perform a phonological analysis of input strings as described in Chapter Two. BITC and WWM do not perform phonological analysis of input strings (I could not determine if SOLAK performed a phonological analysis). Phonological processing may be performed before the input string is analyzed by the parser

(DOLAN, CGP) or during the parsing routine (ALEGRIA). Phonological processing of the input string rewrites that string to reflect an underlying representation of affixes and roots. This allows the parser to match these affixes and roots to its databases and to apply rules that determine the legality of the input string.

3.4.6 Suitability for Field Linguists

A parser for field linguists must meet certain computational and linguistic criteria. The parser should ideally be freely or easily available and platform independent, such as PC-KIMMO. The parser should not require specialized knowledge to run; for example, PC-KIMMO requires the mastery of a complicated rule writing system. The parser should be extensible and modifiable, such as BITC or SOLAK, yet not require programming knowledge to implement. BITC requires Perl programming knowledge to implement. From the standpoint of a field linguist, none of the parsers evaluated are suitable for fieldwork on the basis of computational criteria. PC-KIMMO is the most easily available and requires the least amount of specialized knowledge to implement; however, it is currently unsupported by the developers and the Macintosh version is deprecated. A parser for field linguists must also meet linguistic criteria. The parser should be language independent: only four of the parsers evaluated above are language independent. None of the authors above state whether their parser accepts special characters with the exception of Crane, whose Classical Greek and Italian data include accents and diacritics. On the basis of linguistic criteria, BITC requires the least data encoding: there is no set of rules to add to a parsing routine and the lexicon is not used for parsing purposes. It seems the field linguist must encode his data in a special format if he wants it to be machine-readable: all the parsers require lexical or other databases that are encoded in a special format. Perhaps of the most importance to a field linguist after ease of implementation and fitness to task is the ability to update the parser database automatically such as ALEGRIA or via an easy interface such as BITC's. Only four of the ten parsers include automatic updating or updating via the parser interface. Although PC-KIMMO is deprecated, it is the only parser that a field linguist could download, implement and begin using in the field for morphological analysis.

3.5 Evaluation Summary

In sum, the evaluation of the universal parsers above shows the trend toward TWOL system solutions to morphology parsing as well as a trend toward project specific or commercial applications. Currently there is no free, supported or open-source parsing tool available. The majority of the parsers analyzed above are unidirectional parsers that are either TWOL model-based or function similarly to TWOL systems. As TWOL systems require rules in order to parse, such parsers would be difficult or impossible for a field linguist to implement during data collection.

Furthermore, the majority of the parsers analyzed here apply phonological rules to the input, either through pre-parsing routines or some other mechanism. Again, any parser that requires rules in order to parse would be difficult to implement during data collection. Root recognition remains a parsing concern, as evidenced by the suffixing-only language choice of DOLAN, KOVAL, SOLAK and Qpop. An ideal parser would not be limited to suffixing-only languages. Few of the parsers allow the user to add input during the parsing process or modify the lexicon after implementing the parser, due to the pre-coded nature of finite-state networks. User input during parsing or user modifications to the parser databases after implementation are desirable features.

This evaluation can inform the design of a universal parser intended for the field linguist. The parsers that do not require the user to encode rules or extensive lexicons best suit the needs of a field linguist in the data collection stage of a language documentation project. Although a TWOL system is an ideal computational model of a language, the rules necessary to model the language are complex and therefore difficult to implement for the task of interlinearization. A corpus-based system that relies on frequency data provides the most accurate parsing (CGP can parse large amounts of data accurately); however, without an extensive corpus, listing examples of previous parses allows the user to select the relevant parse (for example, BITC). Data incorporating, order dependent parsing eliminates spurious parses. A modular parser that outputs data in a standards-compliant format would eliminate the problem of legacy data generated by now-unsupported applications such as PC-KIMMO.

4 Parser X Design and Implementation

A successful parser designed for the field linguist must meet both computational and linguistic criteria. To meet computational criteria, it must be open source, platform independent and easily and freely available. Open source and platform independent programs are extensible, allowing the user to modify the software or embed it within other software. The user can contact the developers regarding software problems or improve the software, or request increased functionality from the developers. To meet linguistic criteria, the parser must be language independent, allow user input to the parser databases, and meet current archiving standards. The parser I am proposing, Parser X, meets these criteria and is designed to be maximally useful for the field linguist at the data collection stage of a documentation project.

The problems identified in Chapter Three regarding the use of TWOL system, single-language parsers for data collection preclude basing Parser X on a TWOL model. Instead, Parser X matches string segments to the corpus to build a candidate parse (BITC uses a similar strategy). Parser X's parsing approach is weakly based in Exemplar Theory (Brooks, 1978). Exemplar Theory holds that knowledge is learned over time and updated through experience. In Parser X this is operationalized in that the candidate parse and corpus types returned for a given input form will change over time as new tokens added to Parser X. In Exemplar Theory, when a person encounters something new he first thinks of all the similar things he might have come across in the past. Similarly, in response to an input string, the parser searches for all words with similar affixes. Secondly, the person thinks of a specific thing and draws an analogy between the new object and the specific exemplar in his memory. Similarly, the parser takes from the words with matching affix strings the affix strings with the most frequently appearing matching analysis and builds a candidate parse from these and the root of the input string.

4.1 Parser X Implementation

Parser X is programmed in the Java programming language. The files containing the dictionary and text information are XML files. These choices make Parser X platform independent. Parser X accesses the XML files using Castor and Java files generated by Castor (Guttman et al. 2004). Castor is a data binding framework that allows Java programs to access and modify XML files. The files generated by Castor allow Parser X to easily and quickly access the information in the XML files with a minimum of programming on the part of the programmer. In addition, if changes need to be made to the basic programming of the parser or the encoding of the data it is easy to use the automated tools in Castor to create Java representations of the XML data. Parser X is therefore platform independent, extensible, and modifiable by a user with programming knowledge. The parser can be freely obtained from the author and requires no programming knowledge to install and run. Thus, Parser X meets the computational requirements of the field linguist.

To implement Parser X, I used Upper Necaxa Totonac data collected by David Beck for the Upper Necaxa Totonac Project. This data is stored in a database: a large text file of Upper Necaxa Totonac language data (a dictionary and corpus of texts) encoded in Extensible Markup Language (XML) and an XML Schema file that describes the encoding of that data. These files were designed and created by David Beck. I used transformation files to re-encode the database using my own simplified schema. This re-encoding eliminated approximately one third of the data; for example, most metadata associated with each dictionary entry or text such as speaker name, recording file name or date of collection was eliminated. Although this data is important, it is not used by the parser.

4.2 Parser X Design

The parser design is modular. In order to implement the parser for a particular language, the user adds a number of XML files: a dictionary file, an affix file, and an index file. The schema files for all XML-encoded files are found in Appendix B. The index file is a list of corpus tokens with their frequency data created from the user's corpus with a helper application called Indexer. Once these files are added, the application can be run. New files can be appended to the corpus and Indexer can be re-run to update the index file. The dictionary and affix files must be manually encoded and updated. Figure 4-1 illustrates the basic design of Parser X.

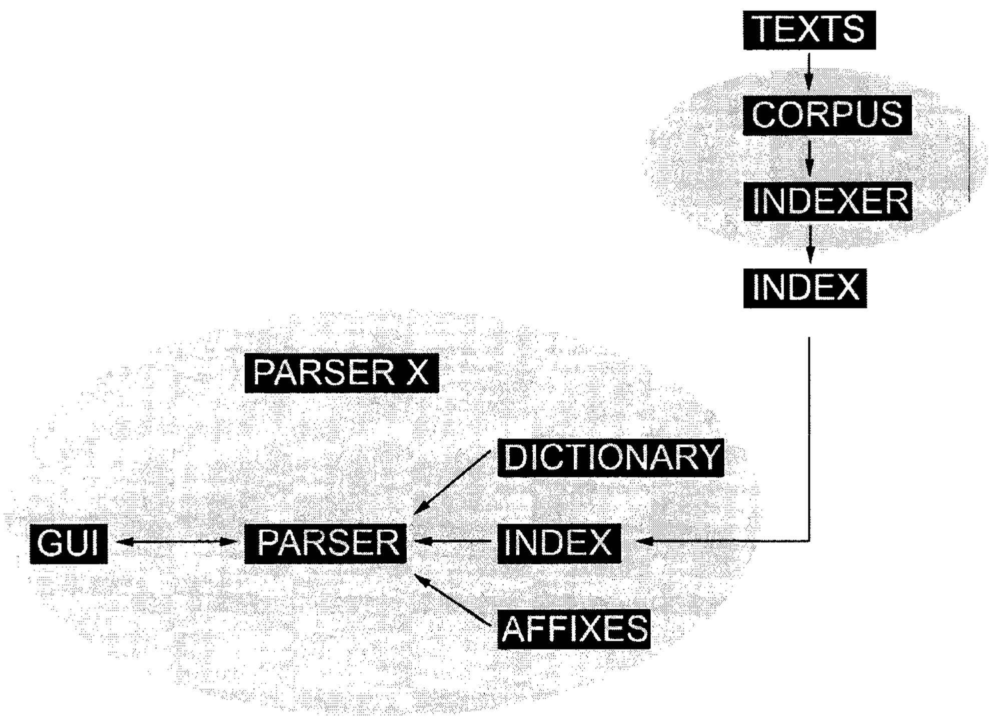


Figure 4-1. Schematic of Parser X Design.

This schematic shows the relationship between two applications, the Indexer application and the Parser X application. The user creates a corpus file from the texts and adds it to the Indexer application folder. The Indexer application generates the index file. The user adds the index, dictionary and affix files to the Parser X folder. Parser X is comprised of these files, the parser code and the graphical user interface (GUI) generated by that code. The Parser X code and files are described in detail below. Note that the Indexer application generates the index file, but that the index file must be manually added to the Parser X application folder by the user.

4.2.1 Texts and Corpus

The corpus is a re-encoding of three texts from the Upper Necaxa Totonac database. Although these texts are in XML format, plain text versions were used to create the corpus. The texts were recorded in the field and transcribed by a native speaker aided by a trained linguist. Each line of the texts is written out in a standard orthography and interlinearized. An excerpt of one of the texts used to create the corpus is shown in example (11).

(11) Totonac

nakintamaxkí:	la'hatín	ixkawa:yúj
na-kin-ta-maxkí:	la'ha-tín	ix-kawa:yúj
fut-1obj-3pl.subj-give	cls-one	3po-horse

'they are going to give me one of their horses'

In example (11), the first tier is in a standard orthography designed by David Beck. The second tier is a morphological breakdown written out by the linguist performing the transcription. The third line is the morphological tier that uses both conventional abbreviations for morphemes and novel abbreviations for those morphemes that have no conventional abbreviation in the literature. The final line is a gloss in English. Regardless of the format of the original texts, the Indexer requires a corpus that is specifically formatted in accordance with rules set down in an associated schema document. The corpus is used by the Indexer to generate the index file.

4.2.2 Indexer

In order to accelerate processing, the corpus is indexed using the Indexer. The Indexer is currently a command line tool; however, in future releases it will be embedded in the parser.

The Indexer requires the corpus be formatted in XML. For example, the text excerpt in example (11) is converted to the XML code shown in example (12).[14]

(12)

```
<Lineblock>
        <Line><w>nakintamaxkí:</w><w>la'hatín</w><w>ixkawa:yúj</w>
        </Line>
        <Mrph><w>na+kin–ta+maxkí:</w><w>la'ha+tín</w>
                <w>ix+kawa:yúj</w></Mrph>
        <IG><w>FUT+1OBJ+3PL.SUBJ+give</w><w>CLS+one</w>
                <w>3PO–horse</w></IG>
        <Gloss><w>they</w><w>are</w><w>going</w><w>to</w>
                <w>give</w><w>me</w><w>one</w><w>of</w>
                <w>their</w><w>horse></w></Gloss>
</Lineblock>
```

This excerpt is then indexed using Indexer into the index fragment in example (13).

[14] Although the Upper Necaxa Totonac XML database designed by David Beck has XML-encoded texts, I manually encoded the plain text versions of the texts into XML. Theoretically it is possible to transform the original XML texts into the format required by the indexer with XSLT.

(13)

```
<?xml version="1.0" encoding="UTF-8"?>
<result lb-count="9" valid="true" xmlns="">
        <lb title-count="0" valid="true" freq="1">
            <iw>FUT+1OBJ+3PL.SUBJ+give</iw>
            <lw>nakintamaxkí:</lw>
            <mw>na+kin+ta+maxkí:</mw>
            </lb>
        <lb title-count="0" valid="true" freq="2">
            <iw>CLS+one </iw>
            <lw>la'hatín:</lw>
            <mw>la'hatín:</mw>
        </lb>
        <lb title-count="0" valid="true" freq="1">
            <iw>3PO–horse </iw>
            <lw> ixkawa:yúj</lw>
            <mw>ix+kawa:yúj</mw>
        </lb>
</result>
```

The <result /> tag contains all the unique words in the corpus. The attributes of the <result /> tag are lb-count, valid and xmlns. The lb-count attribute denotes the number of lines in the corpus; in (13) there are nine lines as indicated by lb-count="9"). The valid attribute indicates that the result file is valid against the schema. The xmlns attribute would indicate a namespace attribute, if there were one. Each of the other tags corresponds to a part of the interlinear gloss in (11). The <lb /> tag refers to a word and its interlinear gloss elements and corresponds to a word within the <w /> tags contained by the <Line /> element in (12). The title-count attribute would indicate the number of the text in the corpus, if more than one text were indexed. Note that the indexer currently only accepts one file. In (13), each <lb /> tag contains an attribute freq: this attribute contains the tally of appearances in the corpus for the word tagged by <lw />. For example, in (13) the attribute tag for *la'hatín* has the value '2' because *la'hatín* appears twice in the corpus. It is only listed once in the index. The <iw /> tag refers to the interlinear gloss of the word contained by a word in the <IG /> element and the <mw /> tag refers to the morpheme-by-morpheme breakdown of the word contained in the <Mrph /> element.

4.2.3 Index

The index is the output of the Indexer. The index lists each word in the corpus (each **type**) along with the morphological breakdown, the morphological abbreviations and the word frequency (number of **tokens** or unique words in the corpus). The index is the list of types from the corpus. These types are the exemplars presented to the user and selected as candidate parses by the parser based on their frequency in the corpus. Against this list, the parser matches substrings of the input form and each token's part-of-speech membership.

4.2.4 Dictionary

The Parser X dictionary is an abbreviated form of the data from the HyperCard Electronic Dictionary by David Beck. The transformation between XML encodings is performed by using an eXtensible Stylesheet Language Transformation (XSLT) processor and an XSLT document that defines the mapping from the tags in the source document to the tags in the target document to generate a file using the target encoding. In the Parser X dictionary, each entry is a headword, part of speech and set of definitions. The dictionary is encoded in XML and validated against an associated schema. Example (14) is an example of a dictionary file showing two entries.

(14)

```
<?xml version="1.0" encoding="UTF-8"?>
<Entries>
        <Entry>
            <Name>a:</Name>
            <POS>adv</POS>
            <Def>over here</Def>
        </Entry>
        <Entry>
            <Name>a:cháj</Name>
            <POS>n</POS>
            <Def>axe</Def>
        </Entry>
</Entries>
```

The dictionary file can contain thousands of entries. The current implementation has approximately 8,000 entries.

4.2.5 Affix List

The Upper Necaxa Totonac database includes a list of affixes. In this implementation of Parser X, the affixes from this list were encoded in an XML file. Example (15) is an example of an affix file showing two entries.

(**15**)

```
<?xml version="1.0" encoding="UTF-8"?>
<affixes>
        <affix>
           <abbr>impf</abbr>
           <type>imperfective</type>
           <name>a</name>
           <position>suffix</position>
        </affix>
        <affix>
           <abbr>each</abbr>
           <type>one by one</type>
           <name>a'</name>
           <position>prefix</position>
        </affix>
</affixes>
```

Each morpheme is encoded by an <affix /> element. Allomorphs of one morpheme are encoded in separate <affix /> elements and there is no indication of a relationship between them other than that they share an abbreviation, type and position. The affixes must be listed in alphabetical order so that the parser will display them in this order in the graphical user interface (GUI). Although the parser displays the affix list, it does not use the affix list in order to parse. The purpose of the affix list is the same as the dictionary's: to allow the user to search for morphemes or roots as an interlinearizing aid.

4.2.6 Parser

The user enters input strings to the parser through the GUI. The parser then searches the dictionary and the index for matching dictionary headwords or words in the corpus; all matches are output to the user. If there are no matches, the parser prompts the user to input a root for the input string in a pop-up window. The parser returns any dictionary matches for the root input by the user, a candidate parse of the input string based on that root and the maximum affix string match in the corpus. The user also has the option of receiving all words in the index whose affixes match those in the input string. Consider the sample parse of the word *milakstín* ('my children') in example (16), below.

(**16**)

```
Input: milakstín
Root entered: lakstín

Output:
          Dictionary matches:

          lakstín (adj) small (plural)

          lakstín (n)
              1. children, offspring, seedlings
              2. (ni) sons/daughters

          Candidate parse:
              milakstín
              mi+lakstín
              2po+ROOT

          Maximum prefix match:
              misandía
              mi+sandía
              2po+watermelon
```

In example (16), the user input the string *milakstín*. After failing to find a match for *milakstín* in the corpus, the parser prompts the user for the root form. If a root is entered that has no match in the dictionary, the parser requests another root. If the user cannot identify another possible root in the input string, the user must cancel the parsing routine. In example (16), the user enters the root *lakstín* and the parser returns the two dictionary entries that match the root. The parser also returns a candidate parse: note that the candidate parse does not gloss the root on the interlinear gloss line, instead giving the term *ROOT*. The dictionary does not contain a one-word gloss as part of each entry to allow the parser to complete the interlinear gloss. In part, this is due to the difficulty in selecting a term that is appropriate for all contexts. The parser has found a match for the root *lakstín*, and continues to the next step. The parser searches for the string *mi* as an <mw /> element, where it would be separated by the symbol '+' from the root: *mi+lakstín*. This is to avoid returning as matches words that contain the string *mi* as part of the root or as a substring of another morpheme. The most frequently appearing word in the corpus that contains the matched affix *mi* is returned with its interlinear gloss as an exemplar for the user. This allows the user to evaluate the candidate parse. In addition, the user sees all the words in the index with matching affixes in the results window. In the above example, *mi* is unambiguous; therefore, the maximum prefix string match is also the only prefix string match. In a case where an affix string is ambiguous, the most frequent instance will be returned as the candidate parse and all other instances are returned as exemplars for the user.

Let us assume that the user is entering words from a text that he wishes to interlinearize. When the user inputs a string to be parsed, the parsing routine follows an algorithm that tries to match the whole input string or substrings of the input string. If no matching word is found in the index, the parser first searches for the matching substring of the prefixes and suffixes along with the user-identified root. In the above example, the parser returned the substring *mi*. The parser sometimes returns an incorrect affix analysis, as shown in example (17).

(17)

Input string: ixchík
Candidate parse:
    ixchík
    ix+chík
    *pst+house

In this case, the correct morphological gloss for *ixchík* is 3PO+house. Due to the homophony between *ix-* '3PO' and *ix-* 'PAST', the parser has two options from which to create the candidate parse. The parser chooses the most frequently-occuring gloss. The user has the option of scanning the other words that have matching affixes but different analyses that are returned in the results. In this case, the parser also returns an example that contains a matching affix with the correct analysis, shown in (18).

(18)

ixpuská:t
3po+woman

The user is then able to interlinearize *ixchík* using the result in (18) as an exemplar.

The parser does not have a rules component to rule out verbal morphology on nouns. The parser returns the most frequently appearing affix match but this is not necessarily the correct affix match. The decision to rely on frequency information can sometimes result in incorrect parses. Any solution to this problem will involve the addition of a rules component. As the goal is to aid the field linguist in interlinearizing data from the beginning of a project when the rules are not yet known, this solution will not be implemented. The user must use the candidate parse, the exemplar parses and the affix table to make decisions about interlinearizations.

If there are multiple affixes in the input string, the parser will return candidates based on matches from the index and dictionary for the root and matches from the index for all recognized affixes. The word *nakila'htzín* has multiple prefixes as shown in Example (19).

(**19**) **Upper Necaxa Totonac**

nakila'htzín
na -ki -la'htzín
'he will see me'

Example (20) shows the parser results for *nakila'htzin* when the root string is entered as *la'hztin*.

(**20**)

```
SEARCH START
          Input: nakila'htzín
          •Dictionary matches: 0
          ◊Corpus types: 0

ROOT ENTERED
          Root entered: la'htzín

          •la'htzín
          vt
          1. see something
          2. guard something, watch over something
          3. pay attention to something, watch out for something
          4. have a certain attitude towards something
          5. treat someone

Candidate parse:
          nakila'htzín
          na+ki+la'htzín
          fut+1obj+ROOT

Maximum prefix match:
          Prefix: naki
          nakili:wá'ya'
          na+ki+li:+wá'+ya'
          fut+1obj+inst+eat+impf:2sg.subj

All 4 prefix matches:
          nakpueblo
          nak+pueblo
          loc+village

          nama'hta'há'lha
          na+ma'hta'há'lh+a
          fut+guard+impf

          nakle:n
          na+k+le:n
          fut+1sg.subj+take

          nakili:wá'ya'
          na+ki+li:+wá'+ya'
          fut+1obj+inst+eat+impf:2sg.subj
```

The candidate parse is built from the matching substring *naki-* and the user-identified root. The parser returns the candidate parse and the exemplar word containing the matching prefix string *naki-* in the prefix match and the maximum prefix match. This word, *nakili:wá'ya'* (21-b), has the same initial prefix string as *nakila'htzín* (21-a).

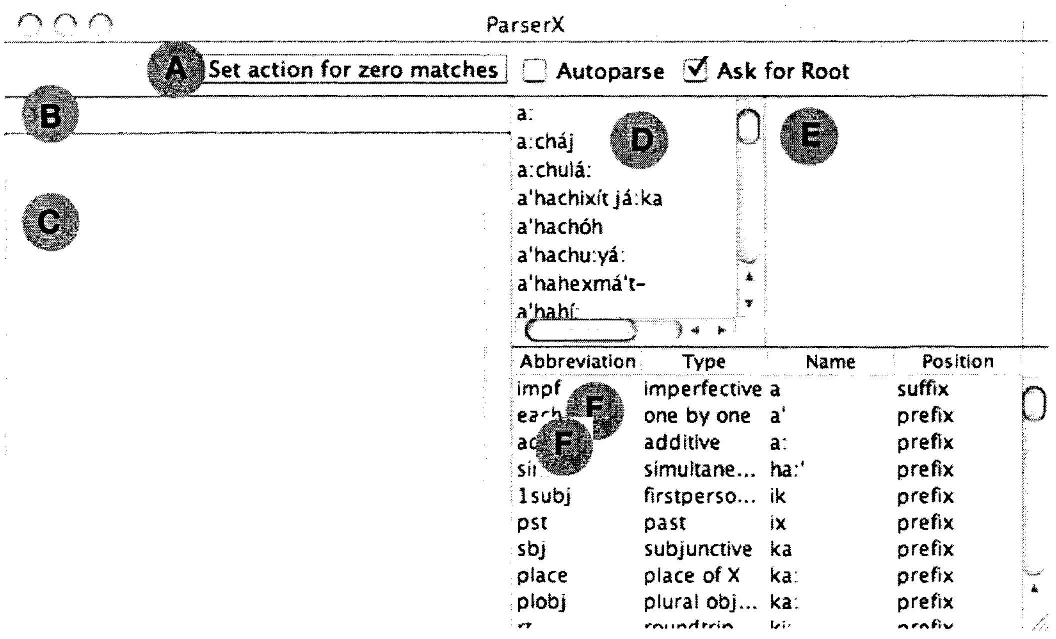
(**21**)

(a)	nakila'htzín na-ki-la'htzín fut-1obj-see 'he will see me'
(b)	nakili:wá'ya' na-ki-li:-wá'-ya' fut-1obj-inst-eat-impf:2sg.subj 'you will eat me'

As the parser returns all unique matches, all words with matching word-initial or word-final segments are included in the results.

4.2.7 Graphical User Interface

The Graphical User Interface (GUI) is the application window that allows the user to interact with the parser. The GUI is shown in Figure 4-2 below.



Set action for zero matches [ ] Autoparse [✓ Ask for Root]

a:
a:cháj
a:chulá:
a'hachixít já:ka
a'hachóh
a'hachu:yá:
a'hahexmá't-
a'hahí:

Abbreviation	Type	Name	Position
impf	imperfective	a	suffix
ea	one by one	a'	prefix
ac	additive	a:	prefix
si	simultane...	ha:'	prefix
1subj	firstperso...	ik	prefix
pst	past	ix	prefix
sbj	subjunctive	ka	prefix
place	place of X	ka:	prefix
plobj	plural obj...	ka:	prefix

Figure 4-2. Parser X Graphical User Interface.

The GUI presents the parser's dictionary as a scrollable list (labeled 'D'). This is the same dictionary that the parser uses to search for roots. The dictionary contains approximately eight thousand entries. If the user selects a dictionary word, that word, its part of speech and definitions appear in a small text area next to the list (labeled 'E'). Each search result remains visible in that scrollable window to allow the user to review his dictionary searches for that session. Below the dictionary is a table of affixes (labeled 'F'). This is merely an *aide-mémoire* for the user, as the affix list is not used by the parser in any parsing routines. The corpus affixes do not include the information about linguistic terms and their abbreviations. As the user identifies the root for the parser, the user can scan the affix list and search the dictionary to determine roots. The GUI has a text field (labeled 'B') for the user to input words to be parsed and an output text area (labeled 'C') for the user to view parser results. Note that the menu bar (labeled 'A') offers the user the option to 'set actions for zero matches.' This refers to the case where the initial input returns no matches from the corpus. Currently, the only programmed option is 'ask for root'; this prompts the user for the root of the input term. The other option, 'autoparse' is currently not programmed.

As described in the beginning of the chapter, Parser X meets the computational requirements of a parser for field linguists. Parser X also meets the linguistic requirements of the field linguist. Parser X is language independent in that it does not require a fixed parsing edge from which to run an analysis routine. Parser X displays all Unicode (UTF-8) characters and all the Parser X databases are in UTF-8. Apart from the database encoding that is outlined in the schema documents in Appendix B, the parser can be used without having to use a special format or learn a set of parser commands. Parser X can be easily updated with new data by appending new texts to the corpus file and re-running the Indexer. After a new index is generated, the user can add this to the Parser X folder. The dictionary and affix files can be directly updated manually by the user. In future versions of Parser X these updates will be possible via the parser interface. Thus, Parser X meets the computational and linguistic criteria required by the field linguist.

5 Use Case Scenarios

The use case scenarios in this section show how different users interact with Parser X to achieve specific parsing goals. The two types of users described in this chapter are field linguists collecting and analyzing data and child language acquisition researchers analyzing data. Both users have similar needs; they need to collect and analyze raw data and ensure that their data is consistently encoded. However, these users also have different needs. For example, the field linguist may be collecting data in a previously undocumented language, whereas the child language researcher most likely already has access to adult language data such as a dictionary and grammar. The field linguist's corpus may be a collection of stories from adult speakers, stories told specifically for the field linguist to record. The child language acquisition researcher's data is a collection of video transcripts: the speakers may be children and adults interacting in a variety of natural contexts. The child language researcher therefore needs access to a video record as well as to an audio record in order to analyze his data. It is also particularly important for the child language researcher to encode his data consistently because child language data contains utterances that can be transcribed or analyzed in different ways and the analysis of the data depends on a consistent analysis of these utterances. While both users are creating interlinear glosses for their texts, the user's needs are different.

5.1 *First Scenario: Field Data Interlinearizing Aid*

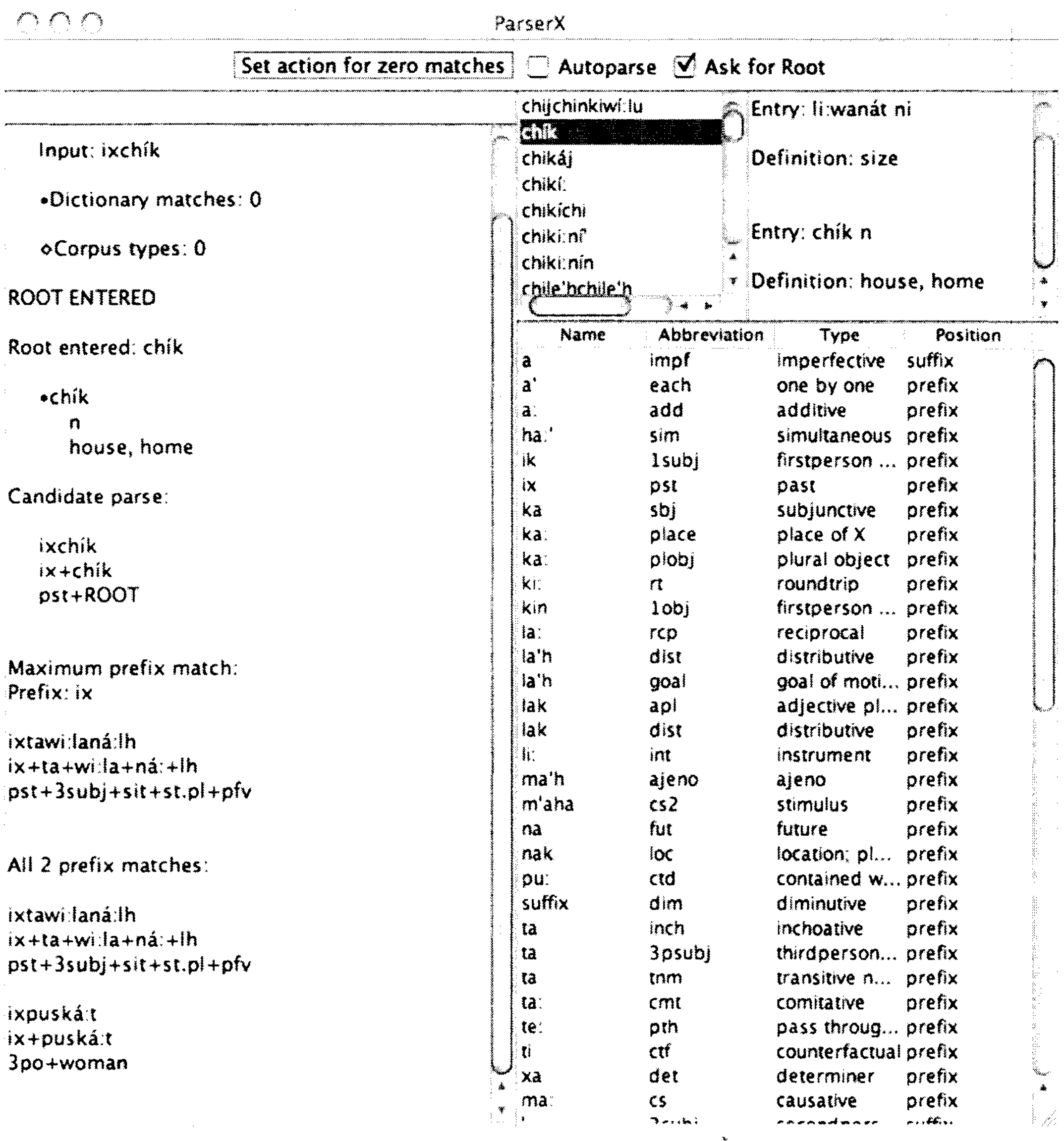
The primary use of Parser X is to aid the field linguist in interlinearizing corpus texts. After texts are recorded, transcribed and translated, the linguist can interlinearize that text by adding a morphological analysis. In order to add an interlinear gloss to a text or text fragment, the linguist must refer to a set of resources: a list of affixes, a set of morphological paradigms, a grammar and a dictionary. Often, the linguist also reviews the audio or video recording from which the transcript is generated. The linguist glosses each word individually, in sequence, until the end of the document is reached. This practice can result in inconsistent encoding or the perpetuation of errors in orthographic representation. As Parser X can return all parses of a particular word and all parses of similar constructions, the use of Parser X will result in more consistent encoding by allowing the linguist to review previous analyses of similar strings.

In this use case scenario, a field linguist must interlinearize the transcript of an interview conducted between two native speakers in the field. This interview is a plain text document and the user is interlinearizing it using Parser X and a text editor. The user has an audio recording of the interview in case he needs to verify a particular word.

In order to implement Parser X in his target language, the linguist must create the dictionary and index files. The linguist is part of on-going research, therefore a substantial dictionary is already complete and several texts are already completely interlinearized. First, the linguist must re-encode the corpus by transforming the corpus data into the correct format. In this case, the data is encoded in XML and the user must transform from the original XML encoding into the form accepted by Parser X by mapping the original tag set onto the Parser X tag set in an XSL file and using an XSLT processor (as described in 4.2.4) or by manually re-tagging the corpus. It is important to note that Parser X encoding does not use white space to separate words, it uses the XML tag <w />. After encoding the corpus, the field linguist must generate a corpus index for Parser X by running the command line tool, Indexer. In addition, the linguist must create the dictionary file needed by Parser X by transforming a dictionary to meet the rules set out in the dictionary schema for Parser X (manually or with XSLT). In addition, the linguist must create an XML-encoded list of affixes for the affix table. After creating the index, dictionary and affix files, the linguist can now add these to the Parser X directory. In this example, two fully interlinearized texts comprise the corpus and the dictionary has approximately 8,000 words.

Once the linguist has added the index, dictionary and affix files to the correct directory, he can begin to use the application. The example of use in this scenario is the addition of an interlinear gloss to an interview transcript. In order to effectively add a gloss using Parser X, the linguist also needs a text editor to create the transcript document and a media player to listen to the original recording. He listens to the interview as he interlinearizes the transcript. The linguist interlinearizes each word in the text individually. After the interview is interlinearized and encoded in XML, the linguist appends it to the corpus file and runs the Indexer to re-index the corpus. Ideally, in future versions of Parser X the Indexer will be integrated into Parser X, allowing the user to update the index while the parser is in use.

The text resources offered by Parser X are open in that the dictionary, affixes table and corpus can be updated by the user. Another benefit of using Parser X as an interlinearizing aid is that it allows the user a dynamic view of affixes, corpus search results and dictionary entries. If the user does not recognize an affix or is having trouble isolating the root in the input form, he can scroll through the list of affixes to find more information about the affixes in the target language. If the user does not accept the candidate parse of the input string, he can scroll through the list of words that contain matching affix strings in the results window. If the user prefers to look up roots in the dictionary before inputting the root string to the parser, the user can scroll through the dictionary or scroll through the dictionary search results window to look at the results of previous dictionary searches. Parser X incorporates these text resources in a simple viewer. Figure 5-1, below, is a screenshot of Parser X in use.



Autoparse [ ]   Ask for Root [✓]

```
    Input: ixchík

    •Dictionary matches: 0

    ◇Corpus types: 0

ROOT ENTERED

Root entered: chík

    •chík
       n
       house, home

Candidate parse:

    ixchík
    ix+chík
    pst+ROOT


Maximum prefix match:
Prefix: ix

ixtawi:laná:lh
ix+ta+wi:la+ná:+lh
pst+3subj+sit+st.pl+pfv


All 2 prefix matches:

ixtawi:laná:lh
ix+ta+wi:la+ná:+lh
pst+3subj+sit+st.pl+pfv

ixpuská:t
ix+puská:t
3po+woman
```

```
chijchinkiwí:lu
chík
chikáj
chikí:
chikíchi
chiki:níʔ
chiki:nín
chile'hchile'h
```

```
Entry: li:wanát ni

Definition: size


Entry: chík n

Definition: house, home
```

Name	Abbreviation	Type	Position
a	impf	imperfective	suffix
a'	each	one by one	prefix
a:	add	additive	prefix
ha:'	sim	simultaneous	prefix
ik	1subj	firstperson ...	prefix
ix	pst	past	prefix
ka	sbj	subjunctive	prefix
ka:	place	place of X	prefix
ka:	plobj	plural object	prefix
ki:	rt	roundtrip	prefix
kin	1obj	firstperson ...	prefix
la:	rcp	reciprocal	prefix
la'h	dist	distributive	prefix
la'h	goal	goal of moti...	prefix
lak	apl	adjective pl...	prefix
lak	dist	distributive	prefix
li:	int	instrument	prefix
ma'h	ajeno	ajeno	prefix
m'aha	cs2	stimulus	prefix
na	fut	future	prefix
nak	loc	location; pl...	prefix
pu:	ctd	contained w...	prefix
suffix	dim	diminutive	prefix
ta	inch	inchoative	prefix
ta	3psubj	thirdperson...	prefix
ta	tnm	transitive n...	prefix
ta:	cmt	comitative	prefix
te:	pth	pass throug...	prefix
ti	ctf	counterfactual	prefix
xa	det	determiner	prefix
ma:	cs	causative	prefix

Figure 5-1. View of Parser X in Use

Parser X's GUI window can be resized by dragging on the bottom right corner to best fit into a desktop environment. For example, the Parser X GUI, a text editor and an audio or video player can all be running and visible to the user on the screen at the same time.

The parser's search results display a variety of information. For example, the first search result is the best candidate parse based on matching prefixes and suffixes in the corpus. Subsequent results include the corpus type with the longest matching prefix and suffix and all results with unique prefix and suffix matches. By reviewing unique prefix and suffix matches, the user can avoid coding inconsistencies. Coding inconsistencies can be difficult to find in a large corpus without a concordance view or a fuzzy search function (for example, one that ignores accents). In Parser X, the search results include words with matching affixes if their interlinear glosses for those affixes are different from each other, making the inconsistency of encoding an affix with more than one morphological gloss more evident. However, Parser X is not designed to catch all coding inconsistencies, for example if words with the target affix strings contain misspelled affix strings or affix strings that are altered by phonological processes they will not be matched and therefore not returned in search results. Parser X could be enhanced by fuzzy searching to allow the user to see, for example, all unique affixes encoded as third person plural.

5.2 Second Scenario: Child Language Acquisition Researcher

In this scenario, a child language researcher with CHAT encoded transcripts of child language recordings is using the CLANX text editor and Parser X to analyze his data using the CHILDES system of analysis. Parser X contributes to the interlinearization process in three ways. Primarily, Parser X aids the linguist by performing the type of morphological analysis described in the first use case scenario. This usage has been described above in detail and I describe it briefly in this new context. I describe in detail how Parser X eliminates one type of CHAT encoding inconsistency and aids the linguist in eliminating another type of coding inconsistency.

In order to explain how Parser X aids the child language researcher, first it is necessary to understand why this researcher might choose Parser X to perform the morphological analysis. In this scenario, the researcher has an adult language corpus on which to base his analysis, a child language corpus he is in the process of analyzing, a dictionary and an affix list for the target language, Upper Necaxa Totonac. He is using the CHILDES system to interlinearize his database of transcripts. The CHILDES system is described on the CHILDES Web site as a system of tools for the study of conversational interactions. This system consists of a database, CHAT encoding and CLANX analysis. CHAT is a data archiving and encoding standard that has been used by child language acquisition researchers for many years. The user annotates a CHAT-encoded transcript using CLANX, a text editor and data analysis application developed specifically for the CHILDES system. Users can download CLANX as it is a freely available and multi-platform text editing and analysis program. CLANX ensures a strict adherence to the CHAT encoding standard. Unlike Parser X, CLANX does not incorporate a dictionary or a corpus as an aid in interlinear glossing. CLANX, however, does have a built-in parser called the MOR module.

Despite MacWhinney's claims that the MOR module of the CLAN analysis engine is language independent (2000, p. 113), the early programming focus on English limits the parser's effectiveness for languages with more complex morphology. In CLAN, all language-specific information is isolated in a series of files (2000, p. 113). However, the basic system underlying the MOR module originally listed the rules for English (Hausser, 1989, p. 2). Since English has a limited number of inflectional morphemes, the file sizes were small and did not significantly affect processing speed. For a highly inflected language, the file size would be large and impractical. Without modification to the parser structure, the MOR module of the CLAN engine is restricted to parsing analytic, isolating languages. In addition, the MOR module requires that the user create a rules file for the target language. If the child language acquisition researcher wants to analyze his data, CLANX does not provide a well-designed tool for languages with complex morphology. The user can perform morphological analysis of his transcripts using Parser X with an adult corpus index while interlinearizing his child language transcripts in the CLANX text editor.

Before the child language acquisition researcher can begin using Parser X to interlinearize data using his child language transcripts as the corpus (or as part of the corpus), he must re-encode his data in the Parser

X format. The user can easily encode his CHAT transcripts in Parser X format because the two encoding formats have a one-to-one relationship between elements, with the exception of the <w /> elements which can easily be inserted using a search-and-replace function. Manually performing this encoding should be straightforward. Example (22) shows an excerpt of CLANX encoded data file YON220704.CHA (collected and transcribed by Vianey Varela under the direction of David Beck and Johanne Paradis for the Upper Necaxa Totonac Project and translated by Catalina Fuentes-Muñoz):

(22)

```
*CHI:    naklakpusa: kilá ja:tzé
%bre:    na-ik-?? kilá ja:tzéj
%mor:    FUT-1SUBJ-VT|?? POS|mine ADJ|bad
%spa:    lo voy a hacer pedazo el mío no sirve
%eng:    I'm gonna break mine in pieces, it is bad
%tim:    25:04
*CHI:    ay
%bre:    ay
%mor:    INTJ|ay
%spa:    ay
%eng:    ay
%tim:    25:26
*SIS:    bebé tu:chu:
%bre:    bebé tu:chu:
%mor:    N|baby PRN|what_thing
%spa:    bebé ¿qué cosa es?
%eng:    baby, what is this?
%tim:    25:38
```

This encoding corresponds to Indexer corpus encoding in the following way:

- Each utterance and its associated tiers correspond to a <Lineblock /> element.

- *CHI and *SIS denotes the speaker of the line block and introduces the line. This corresponds to the <Line /> tag.
- %bre denotes the morphological breakdown of the line. This corresponds to <Mrph /> tag. All hyphens on this tier must be replaced by the '+' symbol.
- %mor denotes the interlinear gloss of the line. This corresponds to the <IG /> tag.
- %spa denotes the gloss in Spanish of the line. This line is not used by Parser X.
- %eng denotes it in English. In this example, the %eng tag corresponds to the <Gloss /> tag
- %tim denotes the time of the utterance in the recording; this information is not encoded for Parser X.

The above transcript excerpt in (22), manually re-encoded for use with the Indexer, is shown in (23).

(23)

```
<Lineblock>
        <Line><w>naklakpusa:</w><w>kilá</w><w>ja:tzé</w>
        </Line>
        <Mrph><w>na-ik-??</w><w>kilá</w><w>ja:tzéj<w>
        </Mrph>
        <IG><w>FUT-1SUBJ-VT|??</w><w>POS|mine</w>
        <w>ADJ|bad</w></IG>
        <Gloss><w>I'm gonna break mine in pieces, it is bad</w>
        </Gloss>
</Lineblock>
<Lineblock>
        <Line><w>ay</w></Line>
        <Mrph><w>ay</w></Mrph>
        <IG><w>INTJ|ay</w></IG>
        <Gloss><w>ay</w></Gloss>
</Lineblock>
<Lineblock>
        <Line><w>bebé</w><w>tu:chu:</w></Line>
        <Mrph><w>bebé</w><w>tu:chu:</w></Mrph>
        <IG><w>N|baby</w><w>PRN|what_thing</w></IG>
        <Gloss><w>baby, what is this?</w></Gloss>
<Lineblock>
```

Although the CHAT format appears to easily map onto the Parser X encoding, CHAT allows inconsistencies between the speaker (here, *CHI or *SIS) and *MOR tiers that are not allowed in Parser X encoding.

Running the Indexer on the Parser X encoded transcripts will eliminate the type of data inconsistency allowed in the CHAT format that is not allowed in the Parser X format. In CHAT encoding, the speaker tier that contains the utterance can contain a different number of words than in the associated tiers, such as the interlinear gloss tier. Parser X has a more restrictive encoding format than CHAT. For example, consider the following lines:

(24)

```
*SIS:     juwa:mámankala'htzi
%bre:     ju: wamá: xma:n ka-la'htzí'
%mor:     MTV|look DEM|this ADV|only.OPT-VT|see
%spa:     mira éste nomás míralo
%eng:     look this, just look at it
%???:     check paradigm
%com:     she is showing him a flower
%tim:     25:46
```

Notice that the first line was transcribed as a single string without word boundaries. In this instance, let us assume this is a transcriber error. The user must edit the first line to include word boundaries as shown in Example (25).

(25)

```
*SIS:     ju wa:má man kala'htzi
```

These word boundaries must be consistent with the other tiers for that utterance in the CHAT transcript. The example in (26) is re-encoded manually by the user as the following element <Lineblock /> for the Indexer:

(26)

```
<Lineblock>
        <Line><w>ju</w><w>wa:má</w><w>man</w>
            <w>kala'htzi</w></Line>
        <Mrph><w>ju:</w><w>wamá:</w><w>xma:n</w>
            <w>ka+la'htzí'</w></Mrph>
        <IG><w>MTV|look</w><w>DEM|this</w>
            <w>ADV|only</w><w>OPT-VT|see</w></IG>
        <Gloss><w>mira éste nomás míralo</w></Gloss>
</Lineblock>
```

Although the <Line /> element in (26) appears similar to the %bre tier in (24), the %bre tier cannot be transformed into the <Line /> element unless the user first removes all hyphens between morphemes. The <Line /> element must contain an orthographic representation of the line with each word contained in a <w /> element. If the speaker tier (in (24) the *SIS tier) included all word breaks, it could be transformed into the <Line /> element because it does not contain hyphens. In the CHILDES system, the user is able to create tiers that are specific to her project. In this use case scenario, the %bre tier is used for a type of morphological frequency analysis performed by CLANX that requires the use of these symbols. Normally this type of analysis would be performed on the %mor tier; however, in this use case scenario the researcher has created a new tier that breaks the information on the speaker tier down into morphemes for project-specific reasons. The user must decide which tier will best correspond to the <Line /> element. By running the Indexer on the Parser X encoded transcript, the child language acquisition researcher verifies that each <Line /> element has the same number of words as the <IG>

and the <Mrph /> elements and eliminates the type of inconsistency shown in (24). The Indexer will return the line number, starting at zero, in the original transcript if the number of words between tiers is inconsistent.

(27)

```
Error in Lineblock 0: the line or morph tier differs from the interlinear
gloss tier in word number.
```

After manually correcting any inconsistencies, the user appends the corrected transcript to the existing adult corpus and creates a new corpus index using the Indexer. Although the CLANX parser can perform without this consistency, the inconsistency between the utterance and its analysis must be eliminated or accounted for in a more overt fashion in order to create data that can be shared with other researchers or maintain a digital archive of his transcripts.

A second type of coding inconsistency is the case where the researcher has decided that several words must be transcribed as one word for a specific reason, unlike the case above where we assumed that the utterance was transcribed as one word in error. The child language acquisition researcher can decide that a certain series of words in adult language are one word in child language. For example, the words 'what is it?', in adult speech *tu: chu:*, may be considered one word *tu:chu:* in a child's speech. The researcher may notice that in the child's utterances, *tu:* is always followed by *chu:*, and decide that the child has acquired the 'chunk' *tu:chu:*. If the child were to utter *tu:* in different contexts, then the child can be said to have acquired *tu:*. The user has several coding options for such an instance. One option is to indicate that *tu: chu:* is one 'chunk' by writing it as *tu:chu:*. The researcher purposefully leaves out the white space in the utterance to indicate that *tu:chu:* is one 'child word'. In this use case scenario, the user has decided to start analyzing *tu: chu:* as *tu:chu:* as in (28) from YON220704.CHA.

(28)

```
*SIS:    tu:chu:
%bre:    tu:chu:
%mor:    PRN|what_thing
%spa:    ¿qué es?
%eng:    what is (it)?
%tim:    25:44
```

As *tu: chu:* is now analyzed as *tu:chu:,* the following excerpt in (29) from the file YON220703.CHA now contains a transcription error on the speaker tier (the first tier).

(29)

```
*SIS:    bebé tu: chu: wa:má
%bre:    bebé tu:chu: wamá:
%mor:    N|baby PRN|what_thing DEM|this
%spa:    bebé ¿qué cosa es esto?
%eng:    baby, what thing is this?
%tim:    25:42
```

The instances of *tu: chu:* in transcripts for this child can be corrected using Parser X. This kind of inconsistency is possible with CLANX because in this use case scenario, the first line is not important to any subsequent analysis performed by CLANX. In this case, the student researcher relies on the %bre tier for speaker data as described above. Data encoded with the kind of inconsistency shown in examples (29) is problematic for researchers who rely on data consistency for their analyses, such as corpus linguists. This is one instance of the program influencing the data collection in a negative fashion. The Indexer will reject (29) because the number of words on the <Line /> element is different from the number of words in the <Mrph /> element. Parser X helps the researcher to correct these inconsistencies by finding the tiers with errors in coding. If the researcher changes his criteria of analysis, Parser X can help locate these inconsistencies.

However, the above example of *tu:chu:* illustrates a problematic aspect of CHAT encoding. Let us reconsider the utterance in Example (24).

(30)

```
*SIS:    ju:wa:mámankala'htzi
%bre:    ju: wamá: xma:n ka-la'htzí'
%mor:    MTV|look DEM|this ADV|only OPT-VT|see
%eng:    look this, just look at it
```

In (30), the child language is on the first line and the adult language forms are implied by the morpheme boundaries indicated in the analysis on the subsequent two lines. If the researcher decides that *ju:wa:mámankala'htzi* is a one word utterance that is analyzed as if it were separate words, he will not be able to re-encode these lines in Parser X encoding and run the Indexer successfully to create an index. If the user plans to use Parser X to perform automated parsing, he can use Parser X with an adult-language corpus index to analyze his data and CLANX to maintain his corpus of child language transcripts. As the researcher is using an adult language corpus for the analysis, he must be aware of the adult forms of the words he has chosen to analyze as child-words because he will not find corpus matches for the child words. Once the researcher has successfully run the Indexer on his transcripts, he can add the index to the parser files. After adding the index files to the Parser X folder, he can begin using Parser X to analyze a transcript.

In the case that the user adds his child language files to the corpus, he can use Parser X in conjunction with CLANX to ensure coding accuracy for difficult tokens. In example (9), the user has inconsistently analyzed the form *wa'a:* in the corpus.

(31)

(a) wa'a:
wa'a:
DEM | over.there

(b) wa'a:
wamá:
DEM | this

If the user searches for *wa'a:*, Parser X will return both analyses shown in (9) because it returns all corpus matches and their analyses. Thus, a second type of coding inconsistency can be revealed in the parsing results.

One problem for the child language researcher using Parser X is the lack of metadata for each exemplar. The Parser X schemas do not include any elements that would correspond to linguistic metadata; nor do the Indexer or Parser X include the code to return such data. If the researcher is including in the indexed corpus files from a variety of adults and children or children of different ages or linguistic ability, the researcher might find it important to know which speaker uttered which exemplar. Ideally, future versions of Parser X would include metadata with each exemplar.

Child language data differs from adult language data in that the data contains errors in both competence and performance. The corpus collected by the child language acquisition researcher may contain many variants for words uttered by different children at different stages of development and also by adults. If the researcher were able to search only the index of texts for a specific speaker, the researcher would be able to get search results that include only that speaker's utterances. A feature that allows the child language acquisition researcher to restrict his search to a particular set of texts would allow the researcher to easily search for coding inconsistencies or for changes in the child's speech over time.

Another useful feature for the child language researcher would be a program to transform the CHAT encoded files into XML files that are valid according to the schema used by Parser X. As CHAT encoding is regular and rule-bound, this type of feature would be trivial to implement. If future versions of Parser X were to include such a re-encoding feature, Parser X would become more valuable to the child language acquisition research community. XML-encoded databases allow researchers to easily reuse data and encoding data in XML is a recommended best practice by the Electronic Metastructure for Endangered Languages Data Project (E-MELD).[15]

A tool such as Parser X, that returns all previous parses for a particular input string, enhances consistency for forms that can be analyzed in more than one way. Parser X would benefit from the addition of features tailored to the needs of child language acquisition researchers, such as the automatic re-encoding of CHAT files into Parser X files and the inclusion of metadata for each exemplar. Overall, Parser X is a useful tool for the child language researcher analyzing a transcript using CLANX for two main reasons: it requires XML encoding of the data and encourages consistency of interlinear glosses in the corpus.

5.2 *Evaluation of Parser X*

There are drawbacks to using Parser X in comparison to other software solutions. The most serious drawback is the necessity for encoding all the data in a new format. Every user of Parser X must encode at least three files in Parser X's encoding format described by the schema documents in Appendix B; therefore, this user must have some knowledge of XML, schemas and, if his data is already in XML format, XML transformations. However, this requirement ensures that even a small amount of data conforms to the best practices for digital language data set by E-MELD in that it requires XML-encoded data.

[15] For more information on E-MELD and best practices, see the E-MELD School of Best Practices in Digital Language Documentation Web site.

Parser X's most important asset is that it is open source and freely available. Parser X can be easily obtained and modified by the end user. Open source projects allow a community of users to alter a particular program to meet the needs of the group, to improve upon the original code and to offer support to users. Open source software that retains a user base over time can outlast the original developers and migrate to new platforms. The parsers evaluated above that grew from single programmer projects to commercial projects are no longer available for linguists to use (one exception to this trend are the Xerox tools, which are newly available for limited use via Beesley and Karttunen's *Finite State Morphology*). Parsers such as PC-KIMMO are now legacy software, for example, and the Macintosh versions are no longer supported and do not run on the current Macintosh operating system. Ideally, Parser X will continue to be developed by a community of users.

Parser X meets the design goals set out in the beginning of the chapter by being open source, platform independent and easily and freely available. In addition, the parser is language-independent and allows user input. Furthermore, the parser's data encoding format meets current encoding standards. Parser X is useful for linguists at the beginning stages of data collection and analysis, as described in the use case scenarios in this chapter.

6 Conclusion

6.1 Summary

The goal of this thesis was to discuss the design and implementation of a universal morphological parser for field linguists based on the evaluation of parsers designed for a variety of purposes. I evaluated ten parsers, four based on TWOL models and six that represent a variety of other theoretical models. The evaluation described the success or failure of a variety of parser features and the suitability of different parsing approaches to the parsing needs of field linguists. These needs include computational and linguistic criteria.

The parsers evaluated in Chapter Three are not intended for use by the field linguist and only one of the parsers is intended for language documentation and analysis (BITC). Only two of the ten parsers are easily available for a linguist to install and use, PC-KIMMO and BITC. Unfortunately, PC-KIMMO is not only deprecated but also requires specialized knowledge of TWOL rule writing and BITC requires extensive programming knowledge to implement. Ideally, a parser for field linguists would be easy to acquire and require only a minimum of specialized knowledge to set up and use. Another drawback to the parsers I evaluated is that several of the parsers are language specific (Qpop, DOLAN, SOLAK and ALEGRIA), relying on features of the target language to perform parsing (DOLAN relies on the syllable structure of

Indonesian, Qpop relies on the suffixing-only morphology of Bolivian Quechua). Language-specific parsers are of no use to a field linguist documenting a new language. However, some of the language-specific parsers have desirable features such as automatic updating of new roots (ALEGRIA) or extensible programming (SOLAK) that informed the design of Parser X.

Several design criteria are successfully implemented in Parser X. Like PC-KIMMO, Parser X is easy to freely download and install on a personal computer running the most popular platforms. Like SOLAK, Parser X is also modular and extensible. Furthermore, like some of the parsers evaluated in Chapter Three, Parser X is language independent. Parser X is also standards compliant in that it requires an XML database. Like BITC, Parser X needs only a minimal amount of linguistic data to implement.

Parser X implements the design criteria described in Chapter Two. Parser X is freely and easily available. It can be downloaded from the Parser X Website, installed from the software accompanying this thesis or obtained from the author. Parser X is platform-independent; it can be installed on a personal computer running Windows, Mac OS or Unix. Parser X is open-source; therefore, any user with programming knowledge can modify the source code for his own use or embed the source code in his own application. In addition, the parser does not require programming knowledge to implement. Once the user has added the data files (index, dictionary and affix files) to the parser folder, the parser is ready to use. This step requires a basic knowledge of XML. Parser X is language independent; the parser can be successfully implemented in any language, as it does not rely on language-specific features as part of the parsing routine. Parser X does not utilize a set of morphological or phonological rules, so no rules component is added or derived from the data. This means that unlike most of the parsers evaluated in Chapter Three, Parser X does not have order dependent parsing. Parser X does not perform any type of phonological analysis and is therefore not data incorporating. While both order dependent and data incorporating parsing can improve the accuracy of results, they require rules components and these are difficult to implement without special rule writing routines (like PC-KIMMO), language specific programming (like Qpop) or an extensive rules component programmed by the user (like CGP and Morpheus). Although Parser X does not automatically add new roots to its index file, new data can be added easily to the parser data files when the user updates the corpus and adds a new index to the parser folder. Parser X implements both the computational and linguistic criteria that determine suitability for the field linguist.

6.2 *Future Work*

Parser X's shortcomings, for the most part, can be overcome through future work. The primary shortcoming is the need for the user to re-encode his data in the Parser X format in order to implement Parser X. This requires that the user be familiar with XML. Although XML is recommended for digital language database encoding, not all field linguists are familiar with it. Those who are may not be familiar with XSLT, which is used to map one XML encoding onto another one.

The data re-encoding cannot be avoided, but the difficulty for novice users can be overcome through the addition of a form-style interface that allows the user to input his data directly into the parser GUI. The parser can then create the XML file and add it to the parser files. For researchers using the CHILDES system, Parser X should be able to import and re-encode CHAT documents. The only difficulty in adding such a feature would be in implementing an interface through which the user can correct coding errors in the original CHAT document. Nevertheless, future implementations of Parser X should include interfaces that allow users to input or re-encode their data via the parser. This requires major additional programming.

The data encoding itself must also be changed in future releases to allow the user to access the metadata from the interface. Currently, the XML encoding used by Parser X is purposefully quite simple: the novice XML coder can easily master it. However, this design decision eliminated all metadata associated with the corpus index. This information may be necessary for certain users, for example, if the user wants to know the dialect or speaker of a given exemplar. The database must include metadata for each index item. The user should have the to option to display metadata with each result. Instituting these changes will require new schema documents and minor additional programming.

Another change that will require new schemas and more programming is the needed redesign of the affix table. The affix table should list each morpheme and all the allomorphs of that morpheme. Currently, the only way to indicate the underlying morpheme is to circumvent the encoding system and name each allomorph *morpheme: allomorph* or something similar. The affix table should be modified to allow a new column showing the allomorphs of each morpheme and the columns should be able to be sorted and re-ordered. This requires minor programming and some changes to the affix schema file.

Switching between two applications as described in the use case scenarios can be frustrating to the user. The addition of a text editor to Parser X will increase functionality. The parser will be embedded in a simple text editor that saves files in plain text or XML format. This addition requires major additional programming that will make Parser X much easier to use for interlinearizing texts. An incorporated media player would also enhance Parser X's use to the field linguist. However, by not incorporating a media player, Parser X is not tied to a specific media format. This allows the user a wider choice of encoding formats, as many media encoding formats are platform specific. Furthermore, adding media player functionality would require a substantial change to the programming code of Parser X. There are many free software solutions available to the user for playing sound and video files.

The Parser interface gives the user the option to have each search parsed automatically. However, if the user selects this feature he is informed that the autoparse feature is not complete. When the user enters an input string to be parsed, the parser returns an interlinearization of that string from the corpus index. If there is no match, the user must enter the root segment of the string because the parser does not identify roots. The parsing algorithm that identifies roots efficiently from the corpus or dictionary is complex and has not yet been programmed into Parser X. The algorithm as it is currently designed is not computationally efficient for languages with words of more than 17 characters. Obviously, more work is needed to refine the algorithm before it can be programmed into the parser. Completion of an automatic parsing feature is a priority.

Another serious drawback to Parser X is the long startup time. The startup time from starting the program to being able to use the program depends on the size of the dictionary and index. However, return speed of the results is quite fast. These times are not empirically measured; they are perceptions of the end user. In comparison, commercially available software such as Adobe Photoshop or Microsoft Word has approximately the same loading times. This drawback can be addressed by restricting corpus and dictionary material to domain-specific texts or by having a large database.

In order to make Parser X more useful for field linguists, additional programming is necessary. Yet as it is, the parser described in this thesis has a variety of features that together aid the field linguist in encoding and interlinearizing texts. For example, the parser can be implemented with a small amount of data in a simple XML format. It displays not only a candidate parser but also all results in the corpus index for each input string. This information aids the user in selecting the best candidate parse. In addition, it allows the user to see if there are inconsistencies in previous analyses. It provides the user with a dictionary and a list of affixes with their abbreviations for reference during interlinearization. It is my hope that my design of Parser X itself is useful to other linguists working with endangered language data.

Bibliography

Aduriz I., Agirre E., Aldezabal I., Arregi X., Arriola J., Artola Zubillaga X., et al. (2000, May 31-June 2). *A Word-Level Morphosyntactic Analyzer for Basque*. Paper presented at the Second International Conference on Language Resources and Evaluation, Athens, Greece.

Aduriz, I., Urkia, M., Alegria, I., Artola, X., Ezeiza, N. & Sarasola, K. (1997). A Spelling Corrector for Basque Based on Morphology. *Literary and Linguistic Computing, 12* (1), 31-36.

Agirre, E., Alegria, I., Arregi, X., Artola, X., Díaz de Ilarraza, A., Maritxalar, M. et al. (1992, April 1-3). XUXEN: *A Spelling Checker/Correcter for Basque Based on Two-Level Morphology*. Paper presented at the Third Conference on Applied Natural Language Processing, Trento, Italy.

Agirre, E., Arregi, X., Arriola, J. M., Artola, X., Díaz de Ilarraza, A., Insausti, J. M., et al. (1995, June). *Different issues in the design of a general purpose lexical database for Basque*. Paper presented at the First International Workshop on Application of Natural Language to Data Bases, Versailles, France.

Alegria, I., Artola, X., Sarasola, K. & Urkia, M. (1996). Automatic Morphological Analysis of Basque. *Literary and Linguistic Computing, 11* (4), 193-203.

Alembic Workbench. [Computer program]. Bedford, MA: Natural

Language Group at Mitre. <http://www.mitre.org/tech/alembic-workbench/>

Anderson, S. R. (1988). Morphology as a Parsing Problem. In K. Wallace (ed.), *Morphology as a Computational Problem* (pp. 1-21). Los Angeles, CA: UCLA Occasional Papers in Linguistics, Volume 7.

Antworth, E. L. (1993). Glossing Text with the PC-Kimmo Morphological Parser. *Computers and the Humanities, 26,* 389-398.

Beck, D. (n.d.). Upper Necaxa Totonac Dictionary. Ms, University of Alberta.

Beesley, K. R. & Karttunnen, L. (2001). A Short History of Two-Level Morphology. Paper presented at the 13th European Summer School in Logic, Language and Information. Helsinki, Finland. Retrieved on September 10th, 2006 from the University of Helsinki Web site: <http://www.ling.helsinki.fi/~koskenni/esslli-2001-karttunen/>

Beesley, K. R. & Karttunen, L. (2003). *Finite State Morphology.* Stanford, CA: CSLI Publications.

Bouda, P. & Rempt, B. (2006). "Turkish Examples 2." Kura, A Multi-User Open-Source Linguistic Database. Retrieved March 21, 2006, from <http://www.ats.lmu.de/kura/samples/turkish2.html>

Brooks, L.R. (1978). Nonanalytic Concept Formation and Memory for Instances. In E. Rosch & B. B. Lloyd (Eds.). *Cognition and Categorization*. Hillsdale, NJ: Erlbaum.

Charras, C & Lecroq, T. (1997). Exact String Matching Algorithms. Retrieved April 21, 2006, from <http://www-igm.univ-mlv.fr/~lecroq/string/> Path: Brute Force Algorithm.

CHILDES Web site. (n.d.) Retrieved September 24th, 2006, from <http://childes.psy.cmu.edu/>

CLAN Manual. (2004). Retrieved August 30th, 2004, from <http://childes.psy.cmu.edu/manuals/CLAN.pdf>

Cohen, W. (2004) *Minor Third* [computer program]. Pittsburgh, PA: Center for Automated Learning and Discovery. Retrieved from: <http://minorthird.sourceforge.net >

Crane, G. (1991). Generating and Parsing Classical Greek. *Literary and Linguistic Computing 6* (4), 243-245

Crane, G. (Ed.). (n.d.). *Perseus Digital Library*. Tufts University. Retrieved on September 10th, 2006 from Tufts University Web site: <http://www.perseus.tufts.edu/>

Dolan, W. B. (1988). A Syllable-Based Parallel Processing Model For Parsing Indonesian Morphology. In K. Wallace (ed.), *Morphology as a Computational Problem* (pp. 75-106). Los Angeles, CA: UCLA Occasional Papers in Linguistics, Volume 7.

Electronic Meta-structure for Endangered Languages Data (E-MELD) Homepage. (2006). Wayne State University. Retrieved on April 21, 2006 from <http://E-MELD.org/index.cfm>

E-MELD School of Best Practice: About XML. (2006). E-MELD. Retrieved on September 15th, 2006 from the E-MELD Web site: <http://linguistlist.org/E-MELD/school/classroom/xml/index.html>

Ford, A., Singh, R. and Martohardjono, G. (1997). *Pace Panini.* New York: Peter Lang.

Good, J & Sprouse, R. (2000, December 12-15). SGML Markup of Dictionaries With Special Reference to Comparative and Etymological Data. Paper presented at the workshop on Web-Based Language Documentation and Description, Philadelphia, USA. Retrieved on August 29, 2006 from the University of Pennsylvania Web site: <http://www.ldc.upenn.edu/exploration/expl2000/papers/goodsprouse/GoodSprouse.html.>

Good, J. & Sprouse, R. (2003). *The Berkeley Interlinear Text Collector (BITC).* E-MELD Language Digitization Project Conference 2003. Workshop on Digitizing & Annotating Texts & Field Recordings. LSA Institute, Michigan State University, July 11th-13th. Retrieved on Sept. 15th, 2006 from the E-MELD Web site: <http://www.E-MELD.org/workshop/2003/good-demo.html>

Guttman, W., Visco, C., Joachim, R., Fawcett, A., Synder, B. et al. (2004) Castor. [computer program] <http://www.castor.org/>

Hankamer, J. (1986, June 1). *Finite State Morphology and Left to Right Phonology.* Paper presented at the Fifth West Coast Conference on Formal Linguistics, Stanford, CA.

Hankamer, J. (hank@ucsc.edu). "2 questions re Turkish and keCi." E-mail to Recipient (isabel.klint@gmail.com). 1 May 2006.

Hausser, R. (1989). *Principles of Computational Morphology.* Technical Report, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburg, PA.

Hoijer, H. (2000). *Harry Hoijer's Chiricahua and Mescalero Apache Texts.* Reproduced from Hoijer, H. (1938). *Rpt. of Chiricahua and Mescalero Apache Texts.* Chicago: University of Chicago. Retrieved on April 21, 2006 from the University of Virginia Web site: <http://etext.virginia.edu/apache/>

Ingush Project, The. (2003) University of California. Retrieved on October 30, 2003 from the University of California Web site: <http://ingush.berkeley.edu:7012>

Johnson, C. D. (1972). *Formal Aspects of Phonological Description.* The Hague: Mouton.

Jurafsky, D. & Martin, J. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* New Jersey: Prentice-Hall.

Kahane, A. & Mueller, M. (Eds.). (n.d.). *Chicago Homer.* Retrieved from the Northwestern University Web site: <http://www.library.northwestern.edu/homer/>

Karlsson, F. (1995). Designing a Parser for Unrestricted Text. In Karlsson, F., Voutilainen, A., Heikkilä, J. & Anttila, A. (Eds.). *Constraint*

Grammar: A Language-Independent System for Parsing Unrestricted Text. Volume 4, Natural Language Processing (pp. 1-39). New York: Mouton de Gruyter.

Koskenniemi, K. (1983a). Two-Level Model for Morphological Analysis. In A. Bundy (Ed.), Proceedings of the Eighth International Joint Conference on Artificial Intelligence, (pp. 683-685). Karlsruhe, West Germany.

Koskenniemi, K. (1983b). Two-Level Morphology: A General Computational Model for Word-form Recognition and Production. in *Publication 11* (p. 160). Helsinki: University of Helsinki.

Koskenniemi, K. (1984). A General Computational Model for Word Form Recognition and Production. In Proceedings of COLING-84, (pp. 178-181). Stanford University, California.

Koval, S., Beliaeva, L., Kogan, L. , Mikhailov, A., Nikolaev, V., Piotrowski, R. et al. (2000). Morphological Representation in PC-Based Text Processing Systems. *Literary and Linguistic Computing, 15* (2), 131-155.

MacWhinney, B. (2000). *The CHILDES Project: Tools for Analyzing Talk.* 3rd Edition. Mahwah, NJ: Lawrence Erlbaum Associates.

Mahoney, A., Rydberg-Cox, J. A., Smith, D. A. & Wulfman, C.E. (2000). Generalizing the Perseus XML Document Manager. In *Linguistic Exploration: Workshop on Web-based Language Documentation and Description*. Philadelphia, PA. Retrieved August 21st, 2006 from the

University of Pennsylvania Linguist Data Consortium Web site: <http://www.ldc.upenn.edu/exploration/expl2000/papers/mahoney/mahoney.htm>

Neuval, S. (2002). Whole word morphologizer: expanding the word-based lexicon: a nonstochastic computational approach. *Brain Lang*, *81* (1-3), 454-463.

Neuval, S. & Fulop, S. (2002). *Unsupervised Learning of Morphology Without Morphemes*. Morphological and Phonological Learning: Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON), Philadelphia, July 2002, pp. 31-40. Association for Computational Linguistics. Retrieved on August 30th, 2004 from <http://arxiv.org/abs/cs.CL/0205072>

Oflazer, K. (1996). Error-Tolerant Finite-State Recognition. *Computational Linguistics*, *22* (1), 73-89.

Packard, D.W. (1977). Computer-Assisted Morphological Analysis of Ancient Greek, Proceedings of the 5th conference on Computational linguistics (pp. 343-355), August 27-September 01, 1973, Pisa, Italy.

Parser X Web site. (n.d.) Retrieved September 24th, 2006 from: < http://isabel.klint.googlepages.com/>

Petasis, Georgios (2005). Ellogon. [computer program] Attiki, Greece: Thesi Goritsa Aspropirgou. <http://www.ellogon.org/>

Poibeau, T. (1990). *Bi-directional Automata to Extract Complex Phrases from Texts.* Automata Implementation, WIA '98, Rouen, France. In J.-M. Champarnaud, D. Maurel and D. Ziadi (Eds.) *Lecture Notes in Computer Science, Vol. 1660* (pp. 110-120). Berlin: Springer.

Shoebox 5. Computer Software. (2006). The Linguist's Shoebox. SIL International: Partners in Language Development. Retrieved on April 12, 2006 from <http://www.sil.org/computing/shoebox/>

Simons, G. (1998). The Nature of Linguistic Data and the Requirements of a Computing Environment for Linguistic Research. In Ed. John M. Lawler and H. Aristar Dry. *Using Computers in Linguistics: A Practical Guide* (pp. 10-25). London & New York: Routledge.

Solak A. & Oflazer, K. (1993) Design and Implementation of a Spelling Checker for Turkish. *Literary and Linguistic Computing, 8* (3), 113-130.

Sprouse, R. (2000, December 12-15). Data Types for Interlinear text. Paper presented at the Workshop on Web-based Language Documentation and Description. Philadelphia, U.S.A., Retrieved September 18th, 2004 from the University of Pennsylvania Linguist Data Consortium Web site: <http://www.ldc.upenn.edu/exploration/expl2000/papers/sprouse/interlin_data_model.html>

Sun Microsystems (2003). Javacc Project Home. Retrieved on November 18th, 2004 from <https://javacc.dev.java.net/>

Wallace, K. (1988). Parsing Quechua Morphology For Syntactic Analysis. In K. Wallace (ed.), *Morphology as a Computational Problem* (pp. 145-161). Los Angeles, CA: UCLA Occasional Papers in Linguistics, Volume 7.

Whitehead, D. (Ed.). *Suda On Line.* (2001). Retrieved September 10th, 2006 from the Stoa Consortium Web site: <http://www.stoa.org/sol/>

Appendix A. Abbreviations

1	1st person
2	2nd person
3	3rd person
CAUS	Causative
CIRCUMFIX	Circumfix
DAT	Dative
DEM	Demonstrative
DIST	Distal
DU	Dual
EMPH	Emphatic
FUT	Future
IMPF	Imperfective
INST	Instrumental
INTJ	Interjection
IO	Indirect Object
LOC	Locative
MTV	???
N	Noun
NEG	Negative
OBJ	Object
PASS	Passive
PL	Plural
POLITE	Polite
POSS	Possessive
POT	Potential
PREFIX	Prefix
PRN	Pronoun
PRS	Present
QTV	Quotative
ROOT	Root
SG	Singular
SUBJ	Subject
SUFFIX	Suffix
TRANS	Transitive
V	Verb

Appendix B. XML Schema Files

Corpus Schema: Text1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="Text">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Title"/>
        <xs:element maxOccurs="unbounded" ref="Lineblock"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Lineblock">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Line"/>
        <xs:element ref="Mrph"/>
        <xs:element ref="IG"/>
        <xs:element ref="Gloss"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Line">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="w"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Mrph">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded"
ref="w"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="IG">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded"
ref="w"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Gloss">
    <xs:complexType>
     <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="w"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="w" type="xs:string"/>
</xs:schema>
```

Index Schema: index.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="result">
    <xs:complexType>
     <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="lb"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="lb">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="lw"/>
      <xs:element ref="mw"/>
      <xs:element ref="iw"/>
      <xs:element maxOccurs="unbounded" ref="title"/>
      <xs:element ref="freq"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="lw" type="xs:string"/>
  <xs:element name="mw" type="xs:string"/>
  <xs:element name="iw" type="xs:string"/>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="freq" type="xs:integer"/>
</xs:schema>
```

Dictionary Entries Schema: entries.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="Entries">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Entry"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Entry">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="POS"/>
        <xs:element maxOccurs="unbounded" ref="Def"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="POS" type="xs:string"/>
  <xs:element name="Def" type="xs:string"/>
</xs:schema>
```

Affix List Schema: affixes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="affixes">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="affix"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="affix">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="abbr"/>
      <xs:element ref="type"/>
      <xs:element ref="name"/>
      <xs:element ref="position"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="abbr" type="xs:string"/>
  <xs:element name="type" type="xs:string"/>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="position" type="xs:NCName"/>
</xs:schema>
```