A Morphotactic Infrastructure for a Grammar Customization System

Kelly O'Hara

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

Master of Arts

University of Washington

2008

Program Authorized to Offer Degree: Linguistics

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Kelly O'Hara

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Emily M. Bender

_____
Fei Xia

Date: _____

In presenting this thesis in partial fulfillment of the requirements for a master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purpose or by any means shall not be allowed without my written permission.

Signature_____

Date_____

University of Washington

Abstract

A Morphotactic Infrastructure for a Grammar Customization System

Kelly O'Hara

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

This thesis presents a morphotactic infrastructure created as part of the LinGO Grammar Matrix customization system. The customization system allows users to select options from provided libraries of syntactic and semantic content to create small but well-formed grammars designed as a starting place for development of precision grammars. As many languages have extensive systems of inflectional morphology, it is advantageous to have a general, system-wide method of creating morphological rules. The infrastructure presented here allows morphemes required by the various content libraries to interact properly in terms of morphotactic constraints such as ordering and co-occurrence restrictions. It also allows for the creation of placeholder rules that add morphophonological information but do not contain any syntactic or semantic content. Development was driven by test suites for four typologically and genetically diverse highly inflecting languages. The system showed convergence over these tests, demonstrating that the resulting system is general and flexible enough to used in a cross-linguistic tool such as the Matrix.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I cannot overstate my gratitude to Emily Bender, without whose guidence, support, patience, and optimism this thesis would never have been achieved. I am also indebted all the attendees of the Matrix developers meetings, especially Scott Drellishak and Laurie Poulson for ideas and feedback at all stages of the development of this thesis. Fei Xia's comments and suggestions helped me see this thesis with new eyes and I am grateful for all the time effort she gave to improve the clarity and detail of my writing. I must also thank my husband Ryan for his love and support, which gave me strength through my most frustrated times. I couldn't have done it without you.

This material is based upon work supported by the National Science Foundation under Grant No. BCS-0644097.

Chapter 1

INTRODUCTION

This thesis asks the question of how to provide coverage for morphologically rich languages in a cross-linguistic resource for grammar development. In particular, I concentrate on issues of morphotactics, such as morpheme ordering, morpheme optionality, and inter-morpheme dependencies. I explore how to create a general-purpose morphotactic infrastructure that can be linked into the system's existing analyses of various linguistic phenomena, as well as provide a morphotactic structure as the basis for further grammar development.

The particular grammar tool I focus on is the LinGO Grammar Matrix (Bender et al. 2002). The Matrix is a starter-kit for the creation of broad-coverage precision grammars using Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag 1994). It is presently accessed through a grammar customization system (Bender and Flickinger 2005, Drellishak and Bender 2005), which elicits information from the user-linguist through a typological questionnaire and then outputs a grammar containing the Matrix core grammar as well as additional types, rules, and lexical entries specialized for the language in question. These files are intended as a launching point for further development of the grammar by the user-linguist. These grammars can be loaded into the LKB grammar development environment (Copestake 2002), as well as the PET parser (Callmeier 2000). These systems can (among other things) parse sentences using the rules and constraints defined by the input grammar. While the Matrix core grammar contains a rich set of types supporting lexical rules, the customization system previously did not allow the user to create lexical rules which interact properly with each other.

The goal of the work presented here is to extend the customization system to provide a

robust, language-independent infrastructure for creating morphological rules. The resulting system should take the user's answers to the questionnaire and create language-specific morphological rules that extend the existing type hierarchy of the Matrix core grammar. The morphological system must also create an infrastructure for building rules that will apply morphemes in the correct order, taking into account morpheme optionality and co-occurrence restrictions. The solution needs to be language independent, because the Matrix is a multilingual tool. It must also be independent of any particular syntactic information, because it will serve as a general infrastructure for all the customization system's syntactic libraries, as well as creating placeholder rules for the morphemes for which the Matrix does not yet provide an analysis. The concrete goals of this project are to define the morphotactic phenomena that need to be modeled, define how the relationships between morphemes can be represented in the questionnaire's internal format, and extend the customization system to be able to construct the morphological rule type definitions such that they can be integrated with the core grammar and existing libraries and parse morphologically complex words as the user intended.

Following Bender and Good (2005), morphosyntax and morphophonology are treated as separate systems. The resulting system presented here acts as an independent layer between the morphophonology and morphosyntax. It creates systems of rule hierarchies modeling the interactions between various morphemes and/or morpheme paradigms. Where no syntactic content is available, the system creates a skeleton rule that applies appropriate morphotactic information but does not add any syntactic information. The syntax can be filled in as the user develops an analysis. This also allows new syntactic libraries to be added to the customization system incrementally, without having to disturb already established lexical rules.

I assume that the phonological forms provided in the grammar are regularized. That is, I assume that morphophonological processing will take place outside of this system (beforehand on inputs to parsing, afterwards on outputs of generation). Pre- or post-processing might be done with a finite state transducer, as is common for many morphological pro-

cessing tools. However, FSTs alone would be insufficient for the morphological processing (including morphosyntax) required by the Matrix. Matrix grammars, based in HPSG, produce complex feature structures that encode syntactic and semantic information. Lexical rules add to or modify the features of the lexical entries, outputting more detailed feature structures which can serve as input to further lexical rules, or combined with other feature structures via phrase structure rules. I am attempting to create an infrastructure for these lexical rules. The necessary feature structures are complex, nested structures and could not be reproduced by an FST.

This thesis is divided as follows: Chapter 2 consists of a literature review, describing the Matrix system, the role of morphology in linguistics, and how morphology has been handled in other multilingual grammar tools. Chapter 3 discusses the design goals for this project, both in terms of what linguistic phenomena need to be modeled, and how to do so within the Matrix framework. Chapter 4 describes how the system was implemented and how the design goals were met. Chapter 5 covers the evaluation procedure, and how the system performs on test suites designed for four morphologically rich but genetically unrelated languages.

Chapter 2
LITERATURE REVIEW

In this section I explore the broader intellectual context for the work described in this thesis. In §2.1 I describe the relevant aspects of the Grammar Matrix is greater detail. §2.2 looks at the linguistic topic of morphology and how it relates to this project. In §2.3, I look at how morphology has been addressed in other multilingual resources.

2.1 About the Grammar Matrix

2.1.1 What is the Matrix

Precision grammars are linguistically-motivated rule based grammars designed to model human language as accurately as possible. Unlike statistical grammars, these systems are hand-built by grammar engineers, taking into account the engineer's theory and analysis for how to best represent various syntactic and semantic phenomena in the language of interest. While having a system designed with linguistic precision in mind can help with tasks as wide ranging as testing linguistic hypotheses or machine translation, building such a system is extremely time consuming and labor intensive. In addition, new grammars have typically been built from scratch; prior work tends to be not freely available, or poorly documented, or too specific to the particular language to provide a useful starting point for a new grammar (but cf. Kim et al. 2003). A side effect of this is that precision grammars tend to be substantially different from each other, with no best practices or common representations. This makes it difficult to compare grammars or to use multiple grammars within a larger system. (Exceptions do exist; the ParGram project (Butt et al. 2002) is one example of multiple grammars developed using a common standard)

The Grammar Matrix was designed as a solution to all of these problems. The initial

design goal was to create a language-independent core grammar that could serve both as a seed for new grammars, and as a common development framework (Bender et al. 2002). As development progressed, it became possible to design several libraries that covered more language-specific information (Bender and Flickinger 2005, Drellishak and Bender 2005). For example, there is not one universal strategy for major constituent order, but there is a known range of possibilities. Providing analyses of known strategies for various syntactic and semantic phenomena speeds up the grammar development process by creating a larger and more language-specific grammar from the outset, allowing the grammar engineer to concentrate more on complicated or quirky aspects of the language of interest. The libraries can be accessed via a web interface where the user can select the options relevant to the language they are working on, and the appropriate code is compiled into a starter grammar. As grammars for more languages are developed using the Matrix, there have been improvements to the core grammar. However, most current development is concentrated on the customization system, both in providing more libraries and improving the user interface.

2.1.2 Core Grammar

Matrix grammars are typed feature structure grammars. They consist of a hierarchy of types, which have constraints in the form of feature structures, the elements of which are themselves typed feature structures. A class of types called rules include constraints on the types or features of their daughter(s). The rule must then unify its own constraints with the constraints on the daughter value, with the output being a typed feature structure where the type is the rule type, and the feature structure is the combined feature structure of the rule type and its daughter(s). Grammars produced by the Matrix all include the core grammar of basic types and rules designed as a general-purpose base for all Matrix grammars.

Of particular interest to the work discussed here is the type `lex-rule`. The feature structure for `lex-rule` (1) contains complex features for specifying the syntactic and semantic content of the rules (SYNSEM).

(1)
$$\begin{bmatrix}
\textbf{lex-rule} & \textit{phrase-or-lexrule \& word-or-lexrule} \\
\text{NEEDS-AFFIX} & \textit{bool} \\
\text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CONT} & \begin{bmatrix} \text{RELS} & \begin{bmatrix} \text{LIST} & \boxed{1} \\ \text{LAST} & \boxed{2} \end{bmatrix} \\ \text{HCONS} & \begin{bmatrix} \text{LIST} & \boxed{3} \\ \text{LAST} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
\text{DTR} & \textit{word-or-lexrule \&} \\
 & \boxed{5}\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CONT} & \begin{bmatrix} \text{RELS} & \begin{bmatrix} \text{LIST} & \boxed{1} \\ \text{LAST} & \boxed{6} \end{bmatrix} \\ \text{HCONS} & \begin{bmatrix} \text{LIST} & \boxed{3} \\ \text{LAST} & \boxed{7} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ALTS} & \boxed{8} \end{bmatrix} \\
\text{C-CONT} & \begin{bmatrix} \text{RELS} & \begin{bmatrix} \text{LIST} & \boxed{6} \\ \text{LAST} & \boxed{2} \end{bmatrix} \\ \text{HCONS} & \begin{bmatrix} \text{LIST} & \boxed{7} \\ \text{LAST} & \boxed{4} \end{bmatrix} \end{bmatrix} \\
\text{ALTS} & \boxed{8} \\
\text{ARGS} & \langle \boxed{5} \rangle
\end{bmatrix}$$

The content of these features is unspecified in the basic type. Rules that inherit from `lex-rule` fill in the content of the feature values. Another feature (DTR) specifies the rule daughter, essentially the input of the rule. The daughter is constrained to be either a lexical entry or another lexical rule. Rules that inherit from `lex-rule` can place further constraints on the type of the daughter, as well as the values of the syntactic and/or semantic features. Lexical rules can only have one daughter value. If multiple types could in theory serve as a daughter to the same lexical rule, they need to each inherit from a common

rule type (the Matrix's allowance of multiple inheritance makes this possible). A rule can specify its own type or a type it inherits from as a daughter. It is also possible to get cycles of two or more rules that can feed each other. Infinite loops can be prevented by putting additional constraints on the feature content of the daughter that prevents a rule from applying more than a certain number of times. In the absence of this, the LKB sets a cap on the number of lexical rules that can apply to one word (the default number is 7, but is adjustable), and will stop parsing once this limit is reached.The syntactic and semantic content specified in rules inheriting from `lex-rule` can add or change feature values of the daughter, but no information can be removed.

2.1.3 LKB Processing

Matrix grammars can parse and generate when loaded into a grammar development environment such as the LKB or the PET parser. For this thesis, I used the LKB and concentrated solely on parsing, and so will discuss only these aspects. Once a grammar is loaded into the LKB, the input to parsing is a sentence or sentence fragment. The LKB then applies a bottom-up head-driven parsing strategy to build a feature structure that accounts for every element of the input string.[1] Basic string matching is done to match tokens in the string to lexical entries in the lexicon. Lexical rules (essentially morphological rules) can change the orthographic form of the lexical entries, and so the system also attempts string matching based on the output of lexical rules. Phrase structure rules apply to combine the lexical entries and lexical rule outputs into larger phrases. The LKB continues applying rules until all the elements of the input string are accounted for in one feature structure, or all possibilities are exhausted. All successful parses are returned, and include a phrase structure tree, and the final feature structure which a semantic representation in Minimal

[1]The LKB's built-in efficiency measures include filtering out rules that are guaranteed to fail before attempting unification. See Copestake 2002 for details.

Recursion Semantics (MRS) notation.

When running Matrix grammars in the LKB, lexical rules work as unary productions similar to unary phrase-structure rules. The daughter value is the single input, and the mother is the output. The output contains any additions or changes made by the rule, with the remaining content copied up, as specified in the types it inherits from. The output is the type of the mother. Since the daughter of a lexical rule can only be a lexical entry or another lexical rule, all lexical rules must apply before phrase structure rules apply. The rule can optionally be associated with spelling changes, adding or modifying the orthographic form. Within the core grammar, lexical rules associated with spelling changes inherit from the type `inflecting-lex-rule`, and those that modify feature values without altering the spelling inherit from the type `constant-lex-rule`. These each inherit directly from `lex-rule`. The LKB looks for the particular instances of lexical rule types in two files separate from the main grammar file: one for constant lexical rules and one for inflecting lexical rules. These rule instances inherit from the rule types. For the inflecting rules, the instances are where information such as the orthographic form and whether the affix is a prefix or a suffix is stored.

2.1.4 Customization

The core grammar is designed to be useful for developing a grammar in any human language. However, there are also a number of linguistic phenomena that, while not entirely language-independent, have a known set of possibilities cross-linguistically. Building a library of code providing analyses for e.g. various word order strategies allows the user to simply identify the strategy used in the language of interest and insert the appropriate pre-written code into their grammar. Creating such libraries was a natural step in improving the functionality of the Matrix. A number of these libraries now exist, and more are currently

under development.

Rather than forcing users to find and assemble the appropriate library code manually, the libraries are accessed through a customization system. Users complete an online questionnaire, and a starter grammar is automatically produced based on the options they selected. The customization system consists of three parts: the customization page, the choices file, and the customization script.

Customization Page

The in order for the system to create a starter grammar, the required information must be elicited from the user-linguist. The medium for this elicitation is a web interface.[2] On the main page, the user is presented with an overview of the sections of the questionnaire. As of this writing, the syntax sections consist of: case, word order, sentential negation, coordination, and matrix yes-no questions. Each section consists of a series of questions related to the topic at hand. There is also a section for inputting a basic lexicon. The lexicon requires a noun, a transitive verb, and an intransitive verb. A second noun, an auxiliary verb, and up to two determiners can be added if needed or desired. The user may also enter up to two test sentences, to test the starter grammar's initial functionality.

Choices File

The options selected by the user are saved in a plain text file, called the choices file. This file can be saved independently, whether or not the user builds a grammar from the options. A previously-created choices file can also be uploaded to the customization page, which fills in the customization questions based on that information. The user can then change or add answers as necessary, and build a new grammar from the new choices file.

---

[2] http://www.delph-in.net/matrix/customize/matrix.cgi

The content of the choices file is a list of attribute-value pairs. Although the primary function of the choices file is as input to a computer script, it is also meant to be human-readable. Therefore an effort has been made to make the attribute and value labels transparent and representative, while still remaining concise. In addition, the sections are labeled within the choices file, although this information is not needed or used by the customization script. This allows users to easily find and identify choices without uploading the choices file to the customization page. Here is a sample of the choices file format. The language being described is Zulu (Niger-Congo). The word order is defined as SVO, and the language does not have determiners as independent words:[3]

(2)
```
section=language
language=Zulu


section=word-order
word-order=svo
has-dets=no
```

Before a grammar is built, the choices file is verified to be internally consistent and contain all the information it needs. For example, if the user specifies that negation or question marking is carried only on auxiliary verbs, an auxiliary verb must be provided in the lexicon. Once the choices file is verified, it can then be provided to the customization script to build a grammar.

Customization Script

Matrix grammars are written in a Type Description Language (TDL). The particular language is one developed for usage with the LKB (Copestake 2002), itself based on the syntax

[3]see Appendix A for full choices file for all the test languages

in DISCO/PAGE system (Krieger and Schafer 1994). The customization script is a Python script that reads in the choices file, and uses the information it contains to select or construct relevant sections of TDL code. The output is a collection of files containing the language-specific TDL code. This is then bundled with the core Matrix files to provide a small but functioning grammar fragment.

2.1.5 Summary

The Grammar Matrix is a grammar engineering tool designed to speed up and simplify the development of precision grammars, as well as provide a common framework, making the resulting grammars more comparable. The core grammar provides a language-independent basis for grammar development. The customization system allows users to easily add language-specific code to their grammar from libraries of analyses of various linguistic phenomena. The customization page is a web interface to the customization system. The user's choices on the customization page are recorded in a specialized format in the choices file. This file serves as the input to the customization script, which compiles the relevant code from the libraries and produces the starter grammar.

2.2 Morphology

The heart of language consists of the mapping between forms and meanings. In this context, a morpheme is the smallest unit of form and meaning. While some languages are monomorphemic, with each morpheme functioning as an independent word, most languages have some means of combining morphemes into larger units at the word level as well as at the phrase level. The study of morphology looks at the ways in which languages combine morphemes into larger words. This typically means looking at bound morphemes, which cannot serve as independent words, and how they attach to the stems or free morphemes in

a language.

Of particular interest to this project is inflectional morphology: those affixes which have syntactic function in the grammar. Inflectional morphology is used to encode content such as tense, aspect, person, number, gender, case, negation, discourse status, evidentiality, and so on. While these affixes can carry some semantic content of their own, they exist mainly as a syntactic requirement of the grammar. For example, grammatical gender is assigned largely at random and can vary widely cross-linguistically, both in the genders that are encoded and which gender any particular word is assigned.

In this section, I focus on the role of morphology in the context of multilingual resources. I first discuss why morphological coverage is crucial in the development of a grammar system such as the Matrix. I then set up the differences between morphophonology and morphosyntax, and explain why my system focuses on morphosyntax. Finally, I look at how morphological processing is handled in other multilingual systems.

2.2.1 Why Do We Need Morphology?

Providing coverage for as many languages as possible is an obvious goal for a cross-linguistic tool. The question then arises of what functionality should be developed first. In a resource for building precision grammars made up of detailed feature structures, such as the Matrix, it might seem more worthwhile to focus on providing feature content, and let the grammar engineer work out how the lexical and phrase-structure rules play out in the syntax. In addition, the importance of providing coverage for inflectional morphology may not be immediately apparent to speakers of languages that lack a robust system of inflection.

For many of the world's languages however, inflectional affixes are mandatory as part of even the most basic sentence structures. In previous versions of the customization system,

the user needed to either leave off these mandatory affixes at customization time, or include them as part of the lexical entries. While ignoring the messier aspects of the syntax is expected during customization, we would like the starter grammars to be as linguistically accurate as possible, including being able to parse morphology. For a language with limited morphology, it may be possible to store each inflected word as a separate lexical entry, but some languages have large paradigms and/or many different paradigms, and even if the rules for combining them are straightforward, creating lexical entries for each possible combination is simply unfeasible. Some languages have inflectional systems that allow for repetition or recursion, creating a technically unlimited number of possibilities. For these languages, modeling the morphological rules is necessary if the goal is complete grammatical coverage.

In addition, new content libraries are continually under development. Providing a general system for building morphological rules is helpful for both the Matrix library developers and the users. Instead of writing a morphological rule generator for each new library, developers can instead plug the feature content into the morphotactic system. An independent morphotactic system provides a big-picture view of how the affixes interact with each other in the grammar. While users will still need to provide the customization system with a morphotactic analysis on which to base the rules, the hope is that building the inflectional rule system at customization time will both be faster than constructing the rules by hand, and make the starter grammar easier to understand and develop by the user.

2.2.2 Morphophonology vs Morphosyntax

Morphology interacts with different aspects of a language's grammar. Many phonological rules are specifically based in morphology. An example of this is the English plural /-s/ suffix which becomes [-z] when following a voiced sound. This is change is morphophono-

logical, rather than motivated strictly by phonological conditioning, as evidenced by minimal pairs such as *wins/wince*. Many languages leverage morphology to encode syntactic features, and thus morphology can interact with other aspects of the syntax. The information carried by affixes can correspond to (for example) possible word orders, constraints on arguments or argument structure, the morphology of other words in the phrase, or the semantic interpretation of a phrase. Incorrect affix choice, or omitting affixes altogether can make an entire phrase ungrammatical.

While morphology overlaps with both phonology and syntax, morphophonology and morphosyntax can be treated as distinct topics. If one is interested solely in the phonological processes necessary to render surface forms of a language, the syntactic function of a morpheme is not necessarily relevant. Likewise, if one wishes to focus on the syntax of a language, it is possible to regularize the phonological forms of morphemes of a language (as influenced by, e.g. phonological processes or irregular forms) and look solely at the syntactic function of the morphology. From a syntactic view, using morpheme glosses that don't encode phonological information at all can be perfectly appropriate.

A complete model of morphology of course requires that the morphophonology and morphosyntax interact with each other. Focusing on phonology ignores the important syntactic contribution of morphology, and focusing on syntax can lead to glossed representations that could not be interpreted by a literate native speaker without linguistics training. One solution to this (as discussed in Bender and Good 2005) is to do the two parts as two separate systems, using the output of the one part as the input to the other. The Matrix morphological system is intended as the syntax part of this model.

2.2.3 Summary

Morphology is concerned with how units of meaning combine in a language to form larger words. The information encoded by inflectional morphology varies widely across languages, as do the rules governing how morphemes can attach. The use of inflectional morphology is extremely common cross-linguistically, and many languages have inflectional systems that are either large or governed by complex rules. Modeling these languages as systems of rules that attach morphemes to stems is both more efficient and more linguistically interesting than storing the morphology as part of a lexical entry. Morphology overlaps with both phonology and syntax, and it is possible to look at morphological systems from either a phonological or syntactic perspective. The Matrix and the work discussed in this thesis are focused particularly on morphosyntax.

2.3 Morphology in Multilingual Grammar Tools

There are many systems available to parse and/or generate inflectional morphology, based on both knowledge engineering methodologies (e.g., Beesley and Karttunen 2003) and machine learning (e.g., Goldsmith 2001). There are two main differences between these systems and the Matrix. First, they are not customization systems. Secondly, they are more concerned with morphophonology, while I focus on morphosyntax. There are tools with an interest in morphosyntax, such as Maxwell et al. 2002, but these tools do not provide the in-depth syntactic and semantic analysis found in the Matrix.

2.3.1 XFST

Many tools utilize finite state transducers for morphological parsing. Here I discuss in particular Xerox/PARC XSFT system (Beesley and Karttunen 2003). It consists of two parts: the xfst environment for building and running FST-based morphological parsers, and

the purpose-created lexc programming language, which allows the user to define a lexicon (including sorting lexical entries into classes by inflection patterns) and a series of rules associating orthographic forms of affixes with morpheme glosses. The xfst system can then compile the FST from the defined grammar. Regular expression filters are used to rule out linguistically impossible constructions. The system can generate surface forms from morpheme glosses, as well as glossed forms from surface forms.

While this system is powerful and reliable, it is concentrated mostly on producing accurate string representations, generally over single words. The Matrix builds complex feature structures modeling many interdependent syntactic and semantic relationships both within and between words. This information would be difficult at best to reproduce in an FST environment. However, the two systems are very complementary: the Matrix does not have a robust system for string manipulation, so it is often easiest to work with morpheme glosses rather than attempt to produce accurate orthographic forms. These glossed forms could then be input to an XFST processor, which could produce the surface strings. This has the additional advantage of making the regular expression filters largely unnecessary: many of the filters are used to eliminate strings that are syntactically impossible, which would not be produced or parsed by the Matrix grammars. The Matrix and XFST systems could be seen as each side of the morphophonology and morphosyntax processing, both necessary but working on different aspects of morphology.

2.3.2 SIL Morphological Glossing Assistant

One system that focuses on morphosyntax is the SIL Morphological Glossing Assistant (Maxwell et al. 2002). Marking up interlinear text can be tedious, especially for languages with complex systems of inflectional morphology. Morphological parsers are often used to speed this process up, but many languages have a great deal of homophony in their in-

flectional morphology. If the morphological parser is based solely on morphophonological information, this can lead to spurious parses. Sorting through these parses for the right one can be so time consuming as to cancel out the benefit of using the parser in the first place. Many of these extraneous parses can be eliminated by providing basic morphosyntactic information. Providing this information may be daunting or impossible for people who do not have training in morphosyntactic feature systems.

The Morphological Glossing Assistant (MGA) is designed to aid in the process of glossing interlinear text. It provides a user interface to an ontology of morphosyntactic properties. The user can then select which properties apply to the particular morphemes in the language being documented. It is possible for the user to create their own features as well, as no ontology covers every morphosyntactic feature in every language. The glossing assistant creates feature structures stored in XML format and a general feature system for the language. Morphemes can be assigned features or complex feature structures, and these feature structures can be designated as applying to a particular root type. For example, the feature structure type for a transitive verb could contain a complex feature value corresponding to subject agreement, and another for object agreement. These morphological feature structures contain features designating, e.g. person and number. The feature system provides the possible values for these features. The MGA is designed to be used in conjunction with a morphological parser. It is also intended to exist as part of SIL's larger suite of tools for language description.

This tool is similar to the Matrix, in that it is morphosyntactically motivated, it uses feature structures to model the syntactic content of morphemes, and it provides a means of creating a customized feature system for your language from a language-independent ontology. The most apparent difference between this system and the Matrix is that the MGA is not a grammar tool, but a means of constructing a feature system. It is possible

to create a feature specifying which morphological feature structure can occur to the right of the current one in the orthographic representation, but there is no system of rules for combining the feature structures of various affixes, and no rules for combining words into phrases. And unlike the Matrix, the MGA does not provide any semantic representations.

2.3.3 Expedition

The project most similar to our current work is the Expedition project (McShane et al. 2002). While no longer under development, Expedition was a grammar elicitation toolkit designed to expedite the process of building grammars for machine translation (MT) purposes. Like the Matrix, Expedition relied on knowledge provided by an active user to build the grammars, as opposed to extracting information from a text corpus, for example. However, the design and implementation of Expedition was very different from the Matrix in several fundamental ways.

Expedition grammars are designed only for MT purposes, and can only translate to and from English. Matrix grammars are compatible with the LOGON MT infrastructure (Oepen et al. 2007) and can therefore (theoretically) translate between any language pair. Expedition uses a knowledge elicitation system, called Boas. The target users for Boas are a language informant and software engineer, neither of whom are required to have a background in linguistics or grammar engineering. While this does allow for grammar development in cases where there is not explicit linguistic knowledge available (such as a printed reference grammar), the resulting grammars lack the precision and depth of Matrix grammars. While Boas and the Matrix customization system each automatically generate grammars based on knowledge elicited from the user via a questionnaire, the nature of both the questionnaire and the generated grammars differ significantly because of the difference in target users.

The difference in system design is especially apparent in the generation of morphological rules. Because Boas does not assume explicit linguistic knowledge, it does not require the user to identify stems, morpheme boundaries, or paradigms. Instead, the user tags surface strings with information about the morphological paradigms and morphological rules are generated through a machine learning process. Predicted forms are then presented to the user, who marks them as grammatical or ungrammatical, and this information is fed back into the morphological learner to further refine the rules.

While this system is convenient in the total absence of linguistic knowledge, there are several advantages to assuming expert knowledge in morphological development. Machine learning can be imprecise and arbitrary. Matrix grammars are designed to be precise and linguistically motivated: generalizations and exceptions can be made explicit, and the linguist-user can make decisions about how to model morphological phenomena (such as positing a zero morpheme where there is no phonological material). However, it may be beneficial to develop a hybrid approach drawing on the strengths of both models. Such a system could lower the barrier of entry for would-be grammar engineers while still allowing the user explicit control over the grammars when desired.

2.3.4 Summary

While building multilingual tools that include coverage for morphological phenomena is not a huge field, there do exist relevant systems besides the Matrix. Many systems use finite state transducers to model morphophonological information, but these systems do not encode morphosyntactic information (at least not robustly). The SIL Morphological Glossing Assistant provides a customizable system with morphosyntactic content but no grammar of morphological rules, and the syntactic content is not as rich as that of the Matrix. The Expedition project was a grammar customization system, but the morphological

rules were generated via machine learning rather than explicitly designed by a linguist. Indeed, both the MGA and Expedition were intended to be used by non-linguists, where the Matrix is targeted at linguists.

2.4 Summary

The Grammar Matrix is a tool designed to assist in the creation of precision grammars. The core grammar provides a language-independent base on which to construct the grammars. The customization system allows users to specify phenomena relevant to their particular language, with their selections compiled from libraries of analyses into a starter grammar. Prior to the work described in this thesis, there was not a robust method of including inflectional morphology in the customization system. Providing such a system is important because many languages have inflectional morphology required in even the most basic constructions, but the morphological systems are sufficiently complex that entering each possible combination of affixes as lexical entries is difficult or impossible. Matrix grammars emphasize morphosyntax over morphophonology, and allow morphological rules to be customized and explicitly defined by the user. Some similar multilingual morphological tools emphasize morphophonology or utilize machine learning algorithms, and no other tool provides the rich syntactic and semantic information found in the Matrix.

Chapter 3

DESIGN GOALS

3.1 Overview

As discussed in 2.2.2, the focus of the morphological work in the Grammar Matrix is on morphosyntax, or the way in which morphological processes affect the syntactic and semantic combinatorial potential of the word. Within that broader context, the work presented here focuses on word-level morphotactics, or the constraints on the order and co-occurence of morphemes with words.

3.1.1 Role of Morphotactics Library in Customization

The task at hand is taking users' specifications for how affixes apply in a particular language, and using that information to build lexical rule types that feed each other in the correct order, while taking into account rule optionality and inter-rule dependencies. It is important to keep in mind that this is meant to apply to the current customization system, and so is to some extent limited by the other capabilities of the system. For example, libraries for agreement and for tense/aspect are currently under development, but are not implemented as of this writing. This means that at customization time, inflectional morphology associated with these functions will not be able to add appropriate semantic content, because the customization system doesn't know what that content is. What I am building is essentially a skeleton rule structure: I am putting a framework in place to be filled in by the user as the grammar is developed. As new libraries are added to the customization system, users can link the affixes with their semantic content, and therefore create more fleshed-out

grammars from the outset.

There are several advantages to providing a skeleton rule structure. One benefit is that the starter grammars are more accurate models of language than those produced by the previous system, as the users do not, for example, need to input nouns with case affixes already attached. Another advantage is that users can develop and test whatever morphological rules they wish, without having to work systematically out from the stem. If the user wishes to start by adding content to an affix or affix-paradigm that appears third out from the stem, they can start there without worrying about the semantic content / analysis of the affixes that apply first or second. The skeleton structure also allows users to account for affixes whose syntactic and/or semantic content is not yet covered in the customization libraries. If the rule content for the third affix from the stem can be customized, but the first and second affixes cannot, creating empty rules for the first two affixes allows the third rule to be created such that it will apply in the correct order, at customization time.

Note that these skeleton rule systems will overgenerate because I have not created rules regarding particular paradigm-internal values. While I have rules dictating the morphotactics for an entire paradigm, without any semantic content, it makes it difficult to look inside the paradigms. Though it would have been technically possible to accomplish, it would require a somewhat complex system of additional features and requirements that would become redundant and unnecessary once the user filled in the appropriate content in the feature structure. I decided it would make more practical sense for the user to specify these requirements at the time that they are filling in the rest of the rule content. New rule content libraries should provide the user this opportunity; for phenomena not covered by customization, the rule content can be filled in as the user develops their grammar from the starter grammar provided.

3.1.2 Phenomena Covered

To illustrate some of the morphotactic phenomena that need to be covered, as well as challenges in implementation that need to be addressed, here are examples from the Zulu test suite):[1]

(3) *umu- ntwana u- zo- bon -a in- yoka*
C1- child SC1- FUT- see -FV C9- snake
'The child will see the snake' [zul]

(4) *umu- ntwana u- zo- yi- bon -a in- yoka*
C1- child SC1- FUT- OC9- see -FV C9- snake
'The child will see the snake' [zul]

Zulu has a system of noun classes, which are analogous to grammatical gender. The difference is in scale: rather than two or three classes (e.g. masculine, feminine, neuter), there are about 15 noun classes in Zulu. The C1 and C9 markers in the above example are the markers for the class that the nouns belong to (i.e. 'child' belongs to class 1 and 'snake' belongs to class 9). Verbs mandatorily agree with the noun class of their subject,[2] and transitive verbs can optionally agree with the noun class of their object.[3] This optionality is illustrated in the two examples above. FUT is a tense marker (specifically indicating the remote future). FV stands for "final vowel" and is essentially a thematic vowel, typical of Bantu languages. I will return to this Zulu example throughout this thesis.

---

[1]Zulu examples were constructed using the information in Nyembezi and Doke 1979 and Poulos and Bosch 1997 and have not been vetted by a native speaker

[2]SC1 in the examples above indicate the subject agreement marker. SC stands for "subject concord"; the 1 corresponds to noun class 1

[3]OC9 in (2) is the "object concord" for noun class 9

These examples illustrate several phenomena that need to be accounted for. First, the system needs to be able to attach morphemes only to particular root or stem types. In the Zulu example, the class markers (*umu-* and *in-* in this case) need to only attach to nouns, while the subject and object concords, tense markers, and final vowel can only attach to verbs. The system needs to be able to specify what sort of roots a morpheme can attach to. Some morphemes may also attach to multiple root types, and this needs to be accounted for as well. While the system does not have any preconceived notions of what sorts of inflection root types take, it does need to be able to account for these restrictions in any particular language.

In a highly-inflecting language, most morphemes won't appear immediately preceding or following the root, as there will be other morphemes intervening. Therefore the grammars need to apply the morphemes in the correct order. However, in many languages some morphemes are optional while others are obligatory. In order to accurately model the order in which morphemes appear, the system needs to be able to account for morpheme optionality. In the Zulu example, the object concord attaches directly to the verb, but is optional. The tense marker attaches to the object concord if it is present, but will attach directly to the verb if the object concord has been omitted, or if the verb is intransitive. The system must allow the tense marker to attach in either position.

Other issues arise regarding restrictions and dependencies between various morphemes. This does not come up in the Zulu example, but it was an issue in another test language, Slave (Na-Dene). For example, in Slave, there is one morpheme slot indicating that the subject is plural and a separate, non-consecutive slot indicating that the subject is dual. Both affixes cannot occur in the same word. Since the Matrix applies morphological rules consecutively, each morpheme can only "see" the rule that applied immediately before it. I need to create a way to keep track of non-consecutive requirements, and expose the relevant

information to the rules that need it. So, in the Slave example, there needs to be a way for the dual marker rule to tell whether or not the plural marker has already applied, even if subsequent morphemes have been added after the plural marker. While these restrictions could be caught and rejected by, for example, an XFST system doing morphophonological processing over the output of generation. However, the efficiency of the whole system is improved if these restrictions are modeled in the grammars, so that it is not wasting time on constructions known to be ungrammatical.

I also want to be able to represent zero morphemes, without necessarily forcing the user to explicitly represent them in the orthographic form. That is, I want to provide the option of having a rule apply without modifying the surface string, as opposed to having the orthographic form of a an affix be, for example, '-Ø'. This is especially useful to complete paradigms where one element does not have an overt phonological representation.

3.1.3 Summary

My goal is to create a system that allows a user to customize an infrastructure of morphological rules, as part of a multilingual grammar development tool. This system should tie into the larger tool's syntactic analyses, where possible. Where there is not an analysis provided, the system should build a skeleton rule structure to provide a framework for the user to continue the development of their grammar. This skeleton rule structure should be able to capture the morphotactics of the language, which means accounting for rule ordering, rule optionality, and co-occurrence restrictions.

3.2 Customization Page

As discussed in §2.1.3, the user interface for the customization system is a website known as the customization page. The format of this page is a questionnaire. Options are currently

input in one of three ways: by marking an option by clicking a check box or radio button; by selecting an option from a drop down menu, or by inputting text into a text field (for providing orthographic forms). These selections are then used to create the choices file. This system will need to be updated to accommodate the morphological interface. I have chosen here to focus on the back-end issues of the choices file format and customization script, and leave the user interface for future work. This decision meant I needed to create the choices files discussed in the next section and shown in Appendix A by hand.

However, it is important within this project to keep in mind that the customization page is still the starting point for the customization system. As such, the morphotactics system is somewhat constrained by the information available. For example, the parts of speech provided in the lexicon consist of nouns, verbs, and determiners. Consequently, the system presented here is only designed to handle these sorts of lexical items. It should be trivial to add further lexical types, but types such as adjectives were not explicitly handled or tested.

3.3 Choices File

The choices file is output of the customization page and input to the script that produces the starter grammars. While the user interface for the morphotactics system is yet to be developed, it is still necessary to design a specification for the choices file format. I first need to identify what information needs to be encoded by the choices file.

It is useful in this case to divide the information into two basic types: paradigm information and meta-paradigm information. A paradigm in this context can be thought of as an affix “slot” or position. A paradigm may have many possible values. For example, a paradigm indicating number could have values such as singular, dual, or plural. Other affixes, such as question or negation markers, often only have one possible value, but I still refer to them as paradigms in this context. In the choices file, I need to encode 1) a name

for the paradigm (e.g. “noun class” in the Zulu example) 2) if there is only one possible value, what its orthographic form is, and 3) if there are multiple paradigm-internal values, labels for those values (e.g. the list of noun classes in Zulu) and their orthographic forms.

Meta-paradigm information covers how this paradigm behaves with regards to other affixes and the root. In particular we need to encode 1) if this affix is a prefix or a suffix, 2) where this paradigm can attach (e.g. Zulu’s tense marker can attach to the verb or the object concord), 3) if it is optional or not, and 4) any co-occurrence restrictions (i.e. any morpheme paradigms that must or cannot co-occur with morphemes in the paradigm in question).

3.4 Customization Script

The customization script takes the choices file and uses the information encoded to produce lexical rules. In particular, it needs to take the information about where each morpheme can attach and create a structure of rules that feed into each other in the correct order. Two more specific goals include designating the input and output of a rule as a lexeme or a word, and keeping track of co-occurrence restrictions.

3.4.1 Lexeme or Word

Within the Matrix, lexical entries and the products of lexical rules can be either *lexemes* or *words*. Essentially, a word is well-formed and can be the daughter of a phrase structure rule. A lexeme is not yet sufficiently well-formed and requires additional information (added via lexical rules) before it can interact with phrase structure rules. Words and lexemes are distinguished by a boolean feature.[4] Each rule needs to be designated as a lexeme-to-word (ltow) rule, a lexeme-to-lexeme (ltol) rule, or a word-to-lexeme (wtol) rule. This designa-

[4][INFLECTED +] for words, [INFLECTED -] for lexemes

tion specifies whether, after the application of this rule, the lexical item changes status as to whether it is a well-formed word, or if it requires more inflection to be grammatical. A ltol rule (despite its name) is one whose status does not change between the input and the output. Here is the Zulu example from above:

(5) *umu-ntwana* *u-zo-bon-a* *in-yoka*
C1- child SC1- FUT- see -FV C9- snake
'The child will see the snake'

The lexical entry for *bon* "see" is not a well-formed word. The first morpheme to be applied is the tense marker, in this case the future tense marker *zo-*. However, applying this morpheme does not make the verb well-formed. Because the status didn't change, the rule applying the tense marker is ltol. Ltol rules also include additional morphemes attached to well-formed words. For example, if an additional negation marker is added to the well-formed *u- zo- bon -a*, the input and output are both well-formed words, which makes the negation rule a ltol rule.

The type word-to-word does not exist, as it is functionally the same as lexeme-to-lexeme. In both cases, the word vs. lexeme status is the same in the mother and the daughter. It would be unnecessarily redundant to create one rule type where the input and output were both words, and another where they were both lexemes.

A ltow rule takes a not-fully-formed lexeme as input, and outputs a well-formed word. These are generally identified as the last non-optional morpheme to be applied, but may also be created in conjunction with a wtol rule; see below. In the analysis used in the Zulu grammar, the last non-optional rule for the verb is the final vowel. After the final vowel is applied, the verb is considered a well-formed word by the grammar, and can be used in larger syntactic constructions.

Wtol rules take well-formed words and output lexemes that require more inflection to become well-formed again. They did not exist in the Matrix prior to this project; I found them to be necessary for encoding co-occurrence phenomena, as described below. For wtol rules, three conditions need to be true: 1) The rule is itself optional. 2) the lexical item it attaches to is already considered well-formed by the grammar (either the lexical entry is a well-formed word on its own, or a ltow rule has previously applied). 3) The choices file specifies that if this rule applies, another, subsequent rule must also apply in order for the word to be well-formed. For example, in Slave, the lexical entries are generally well-formed words before the application of any lexical rules. Most lexical rules are therefore ltol rules, as both the input and output are well-formed. However, after the application of an incorporated postposition, the word is no longer well-formed until the application of an affix marking the object of the incorporated postposition.

(6) *be-keh-na-be-ne-w-n-h-tah*
3SG-into-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
'I kick him/her into it (e.g. hole).'

Affixes are applied one at a time starting from the root. In this example, the first affix applied would be the first person singular marker *h-*, followed by the perfective marker, and so on. Until the incorporated postposition *keh-* 'into' is applied, the word is well-formed both before and after the application of each affix. However, if an incorporated postposition is added, it must be be accompanied by a pronominal affix corresponding to its object. Because the affixes are applied from the root out, the object (in this case the third person singular marker *be-*) is applied after the incorporated postposition. If just the postposition is applied, without being followed by its object, the word is no longer well-formed. Therefore, the incorporated postposition rule needs to be a wtol rule: the output of the rule is marked

as a lexeme and a subsequent ltow rule must apply to make the word well-formed again. In this case, the postpositional object marker would be the corresponding ltow rule.

3.4.2 Co-occurrence Restrictions

There are three types of co-occurence restrictions I wish to cover: 1) Affixes that can only apply if a another affix has not applied. I previously mentioned a Slave example where a plural marker cannot occur if a separate dual marker has already applied. 2) Affixes that can only occur if another affix has already applied. In the discussion of word-to-lexeme rules above, I mentioned an incorporated postposition and its object. In this case, the object cannot apply if the postposition has not occurred. 3) Affixes that, if they appear, force a subsequent morpheme to appear. This is the case with the incorporated postposition in the Slave example. If it occurs, its object must also occur. There needs to be special handling of these cases, because these morphemes can be otherwise optional. That is, none of them are needed to create a well-formed word, but the appearance of one can govern the appearance of others.

The difficulty in our situation is that many of these rules will not have any syntactic or semantic content in the grammar as output by the customization system. While I do not intend to look inside paradigms, it is useful for the grammar engineer to be able to account for restrictions over entire paradigms. However, since these rules are not necessarily consecutive, I need to provide a way for the system to keep track of which rules have already applied. In addition, some morphemes can be thematic: required by the syntax but not actually adding any syntactic and semantic content of their own. In Slave many of the verbs have discontinuous stems: in addition to the root word, the stem must have a one or more thematic prefixes that are either meaningless or have become abstracted from the original meaning and are now simply required by the stem. However, other prefixes can occur be-

tween the thematic affixes. For example:

(7) *go-Ø-deeh*
THM-VERB.PREFIX-talk
'talk'

(8) textitgo-h-Ø-deeh
THM-1SG-VERB.PREFIX-talk
'I talk'

The verb prefix is required by all verb roots and can be treated as part of the lexical entry. The thematic prefix textitgo- is required by the lexical entry, but other affixes, such as the subject marker in (8), can intervene between the verb root and the thematic affixes. I did not explicitly handle this phenomenon in my system, as it is lexically defined and not inflectional morphology per se, the machinery for tracking co-occurrence restrictions could also be leveraged to track the occurrence of these semantical empty prefixes in the future.

3.4.3 Summary

The customization script produces lexical rules based on the user's selections in the choices file. At a minimum the implementation should include a means of setting rule types and daughter values such that the rules can apply in the correct order. Two challenges I wish to address in the system are identifying which rules change the status of a lexeme to a word (or vice versa), and keeping track of co-occurence restrictions between non-consecutive morphemes.

3.5 Summary

The customization system consists of the customization page, the choices file, and the customization script. The design goals for the morphological customization are strictly back-end, so the customization page is currently unmodified. The choices file needs to define new types and structure to encode relevant information regarding the content of morphological paradigms, as well as meta-paradigm information regarding the morphotactics. The customization script needs to create lexical rules that apply in the right order based on the choices file. Additional goals for customization include identifying whether or not the rule changes the status between lexemes and words, and tracking co-occurence constraints.

Chapter 4

IMPLEMENTATION

In this section, I present how I met the design goals laid out in Chapter 3. §4.1 describes the choices file format. §4.2 covers the changes to the customization system, by providing an overview of the algorithm followed by more in-depth presentation of each of the design goals of creating intermediate rules, identifying rules as inflecting or constant, identifying lexeme-to-lexeme, lexeme-to-word, and word-to-lexeme rules, and tracking co-occurrence restrictions.

4.1 Choices File

The design goals for the choices file are to encode certain information about the content of the paradigm, and meta-paradigm information necessary to model the morphotactics. I decided to store this information in two separate sections. The first new section of the choices file is the paradigm information, which stores the information about what a paradigm encodes (e.g. case), as well as the individual items within the paradigm (e.g. the specific cases encoded in the language: nominative, accusative, dative, etc.). In the work done for this thesis, the paradigms are not tied to specific syntactic or semantic content, and so the paradigm elements consist solely of orthographic information. Each paradigm has the attribute `name`, the name for the general paradigm. This name is provided by the user, and is intended as a descriptive label, as it will be used as the name for the paradigm supertype rule in the grammar. If there is only one value in this "paradigm" (e.g. a question marker that is either present or absent, but has only one form), the paradigm has an attribute `orth`,

the value of which is the orthographic form for the morpheme. If there are multiple values possible, these are stored in the iterable attribute `aff`, each of which contains its own `name` (again a descriptive label to be used as the rule name) and `orth` values. Here is an example from the choices file for Slave:

(9)
```
pdm2_name=subject
pdm2_aff1_name=1SG
pdm2_aff1_orth=h-
pdm2_aff2_name=1PL
pdm2_aff2_orth=id-
...
pdm16_name=negation
pdm16_orth=du-
```

`pdm2` is the paradigm for subject agreement marking, and is given the descriptive name "subject". There are many values in this paradigm; I have included here the affixes for first person singular and first person plural subjects. `pdm16`, the negation marker, contains one element. Therefore it has no iterable `aff` values and instead has its own `orth` value.

Some elements of a paradigm may not have an overt orthographic (or phonological) form, but still add syntactic information as part of the paradigm. In this case, the orthographic form is specified as 'NONE', which allows the customization script to create a non-inflecting lexical rule.

The second new section is meta-paradigm information, that is, information governing the morphotactics of an entire paradigm. Each meta-paradigm (labeled `morph` in the example below) is associated with a series of attributes, whose values contain information needed to construct the lexical rules. Mandatory attributes are `type`, which specifies which paradigm this morph is associated with (e.g. `type=pdm2` in the same Slave choices file

shown in (7) indicates the meta-paradigm information corresponds to the subject marker paradigm), `aff`, marking whether this morpheme is a prefix or a suffix (not to be confused with the paradigm attribute `aff`), and `opt`, which indicates whether this morpheme is mandatory or optional. The attribute `dtr` corresponds to the lexical rule or lexical type that can serve as input to this rule. Any number of daughters can be listed, but if the possible daughters are due solely to those morphemes being optional, only the outermost possible daughter needs to be listed. Example (8) is a segment of the Zulu choices file. `pdm3` marks the object concord, which can attach to transitive verbs (as intransitives will not have an object to mark), and `pdm4` is the tense marker which can attach to either an intransitive verb or to the object concord. The object concord is optional, and so the tense marker can also attach directly to transitive verbs, but the customization script is designed to discover and account for this optionality (See §4.2).

(10)
```
morph2_type=pdm3
morph2_aff=prefix
morph2_opt=yes
morph2_dtr1_type=tverb
morph3_type=pdm4
morph3_aff=prefix
morph3_opt=no
morph3_dtr1_type=iverb
morph3_dtr2_type=morph2
```

Optional leaf values not illustrated here include `forces`, which indicates that a subsequent rule must be applied if this rule applies, `req` which indicates that this rule can only apply if a previous, non-consecutive rule has already applied, and `disreq`, which indicates that this rule can only apply if a previous, non-consecutive rule has not applied.

4.1.1 Summary

The choices file needs to encode the information about inflectional paradigms and their morphotactics in attribute-value pairs, for use by the customization script. For the paradigm information, I create a series of embedded values containing labels for the overall paradigm, and the labels and orthographic forms for affixes contained in this paradigm. For the meta-paradigm (morphotactic) information I create a series of attributes encoding the optionality, affix type, daughters, and co-occurrence restrictions associated with a particular paradigm.

4.2 Customization Script

Once the choices file is complete, it can be processed by the customization script. In this section I will look at an overview of the general algorithm used to create the morphological rules, followed by discussion of how various design goals were implemented in the system.

4.2.1 Algorithm Overview

To implement the morphological infrastructure, I needed to write additional functions for the customization script to handle the updated choices file information. Here I discuss the main algorithm for creating morphological rules, CUSTOMIZE_INFLECTION(), and the function FIND_DAUGHTERTYPE(). Pseudocode for the main algorithm CUSTOMIZE_INFLECTION() is given in (11) and for FIND_DAUGHTERTYPE() in (12). Additional functions are discussed in the subsequent sections, as laid out below. Some helper functions, for example one that finds what lexical types serve as the roots that an affix indirectly attaches to, are not shown.

The basic algorithm is simply a while loop over the `morph` entries in the choices file. Assuming a well-formed choices file, the algorithm will continue iterating over the `morph` entries, and terminate when there are no more.

The rule's DTR value is set by the function FIND_DAUGHTERTYPE(), discussed below.

(11)
```
1  function CUSTOMIZE_INFLECTION():
       tracks ← []
       while CHOICES.HAS_MORE_MORPHS():
          morph ← CHOICES.GET_NEXT_MORPH()
5         basetypes ← FIND_BASETYPES(morph)
          daughtertype ← FIND_DAUGHTERTYPE(morph, basetypes)
          ruletype ← FIND_RULETYPE(morph, basetypes)
          rulename ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(morph)) +
                        '-lex rule'
10        CREATE_BASIC_RULE_TYPE(morph, basetypes, daughtertype,
                                 ruletype, rulename)
          CREATE_RULE_SUBTYPES(CHOICES.GET_PARADIGM(morph), rulename)
          tracks.APPEND(CREATE_TRACK_FEATURES(morph, basetypes))
       if tracks:
          ADD_SINGLE_TRACKS(tracks)
15        COPY_ALL_TRACKS(tracks)
```

I then check to see if this rule is a lexeme-to-lexeme, lexeme-to-word, or word-to-lexeme rule in the function FIND_RULETYPE() (15), described in §4.2.3. These types cross-classify with constant and inflecting lexical rules. Assignment of inflecting vs constant and the creation of the rule type definitions occurs in CREATE_BASIC_RULE_TYPE() (16) and CREATE_RULE_SUBTYPES() (19), which are discussed further in §4.2.4.

If the choices file indicates a `req`, `disreq`, and/or `forces` value, that information is recorded by CREATE_TRACK_FEATURES() (20) until all the morphemes have been iterated over. Once all the rules have been created, values for the complex feature TRACK are added if necessary in ADD_SINGLE_TRACKS() (21) or the TRACK feature is copied up wholesale in COPY_ALL_TRACKS() (22). Discussion of TRACK features and co-occurrence restrictions appears in §4.2.5

The first function called is FIND_DAUGHTERTYPE() (12). The lexical rule being created will have a feature DTR, with the value set as the rule type that can serve as input to this rule.

(12)
```
1  function FIND_DAUGHTERTYPE(morph, basetypes):
       if NUM_DAUGHTERS(morph) == 1:
          daughter ← CHOICES.GET_NEXT_DAUGHTER(morph)
          if daughter is a lexical type:
5            daughtertype ← daughter
          else:
             if CHOICES.GET_OPT(morph) == yes:
                nonopt, daughtertype ← INTERMEDIATE_RULE(morph, basetypes):
                if nonopt == False:
10                 for type in basetypes:
                      GRAMMAR.ADD(bt + ':=' + daughtertype)
             else:
                daughtertype ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(
                                                      morph))
       else:
15        nonopt, daughtertype ← INTERMEDIATE_RULE(morph, basetypes):
          if nonopt == False:
             for type in basetypes:
                GRAMMAR.ADD(bt + ':=' + daughtertype)
       return daughtertype
```

FIND_DAUGHTERTYPE() (12) determines what this value is. If the choices file specifies one daughter (line 2), and that daughter is non-optional (line 13) or a lexical type (line 4), then the daughter's type is designated as the DTR value. If the single daughter is optional (line 7) or if multiple daughters are listed (line 14), then it is necessary to create an intermediate type that all possible daughters can inherit from. The function INTERMEDIATE_RULE() is shown in (13) and discussed in §4.2.2.

4.2.2 Intermediate Rule Types

The Matrix only allows one daughter value to be listed for each rule. If there is only one daughter listed in the choices file, and it is either a root type (e.g. noun) or a non-optional inflectional rule, then that is the only possible daughter. If there are multiple

daughters listed, or the single listed daughter is optional, then I need to provide a way to designate all these daughters as being of a single type. This is accomplished through multiple inheritance. The system creates an intermediate type in the hierarchy and have all the possible daughters inherit from this type. That type is then designated as single daughter value for the rule in question. When creating intermediate rules for daughters that are optional, the script traverses the tree of possible rule orderings recursively, based on the choices file. The daughters and the daughters' daughters all inherit from the intermediate type, stopping after reaching a root or non-optional morpheme. This feature allows the user to only specify the outermost position where the meta-paradigm in question can appear, and not have to explicitly list all the possible morphemes it can appear next to once rule optionality is accounted for. Pseudocode for INTERMEDIATE_RULE() appears in (13).

For an example, I return to the optionality of the object concord in Zulu. The outermost affix the tense marker can attach to is the object concord. However, the object concord is optional, and does not apply at all in the case of intransitive verbs. Therefore, the rule needs to specify that the daughter can be either the object agreement (OC for object concord) or a verb root. To illustrate how this function works, I will walk through an example using the choices file fragment from example (8), beginning with `morph3`.

When the function first gets called, it is called with the arguments `(morph3, dtr=null, depth=0)`. Since the `dtr` value is null, the first step is to create a dtr type (lines 2-5). The function CHOICES.GET_NAME() takes the morph, looks at its associated paradigm value, then looks at that paradigm and returns the top-level name. In this case the name is 'tense' (This is not shown in the choices file fragment, but can be seen in the full choices file in the appendix.) The algorithm creates a type called `tense-rule-dtr`.

`morph3` is not a lexical type (line 6), so the function moves on to look at the daughters (line 11). `morph3` has two `dtr` values: `iverb` and `morph2`. `iverb` is a lexical type, so

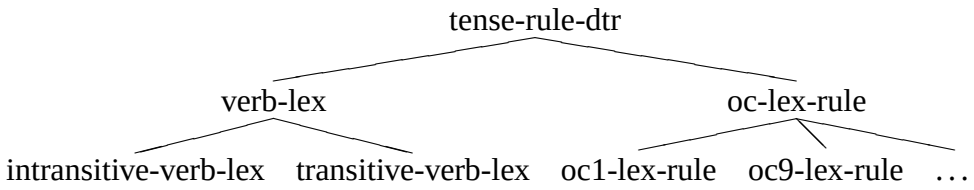
(13)
```
1  function INTERMEDIATE_RULE(morph, basetypes,
                              dtr=null, depth=0, nonopt=false):
       if not dtr:
          name ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(morph))
          dtr ← name + '-rule-dtr'
5         GRAMMAR.ADD(dtr + ':= word-or-lexrule')
       if morph is a lexical type:
          return nonopt, dtr
       if (depth >0) and CHOICES.GET_OPT(morph) == 'no':
          nonopt ← True
10        return nonopt, dtr
       while CHOICES.HAS_MORE_DAUGHTERS(morph):
          daughter ← CHOICES.GET_NEXT_DAUGHTER(morph)
          if daughter not in basetypes:
             GRAMMAR.ADD(daughter + '-lex-rule := ' + dtr)
15        if CHOICES.GET_OPT(daughter) == 'yes':
             while CHOICES.HAS_MORE_DAUGHTERS(daughter):
                granddaughter ← CHOICES.GET_NEXT_DAUGHTER(daughter)
                if granddaughter not in basetypes:
                   nonopt, dtr ← INTERMEDIATE_RULE(granddaughter,
                                                   basetypes, dtr, depth+1)
20        return nonopt, dtr
```

it has no daughters to check. It will also need to inherit from the intermediate type eventually, but it is added outside of this function. `morph2` is not a lexical type, so its lexical rule needs to inherit from the `dtr` type (line 14). Lexical rule names always take the form of the paradigm name plus '-lex-rule'. In this case `morph2`'s rule is `oc-lex-rule`, which is then set as inheriting from `tense-rule-dtr`. Since `morph2` is optional, the next step is to look at `morph2`'s daughters (lines 14-15), which in this case is just a transitive verb. This is a root type, so no recursion occurs (line 16). The function returns the `dtr` value (in this case `tense-rule-dtr`) as well as whether the last daughter it looked at was optional or not (line 20). Notice that once the `dtr` is created, it is not changed. When the recursion

occurs, its goal is to have each of the intervening rules inherit from the intermediate type.

In the function FIND_DAUGHTERTYPE() (12), each lexical type that is the root type that `morph3` attaches to (directly or indirectly) is set as inheriting from `tense-rule-dtr` as well. This is not done within the intermediate types algorithm, because we want to put the constraint at the highest level possible. In this Zulu example, the intermediate types algorithm saw that the tense affix can attach to intransitive verbs and transitive verbs. However, both of these types inherit from the type `verb-lex`, so I make `verb-lex` inherit from `tense-rule-dtr`, and both transitive and intransitive verbs will inherit this type from the supertype. Similarly, all the particular object concord rule types inherit from `oc-lex-rule`, and therefore also inherit the type `tense-rule-dtr` from their mother. Here is what the type hierarchy then looks like:[1]

(14)



4.2.3 Word vs. Lexeme

As previously discussed, a "word" in the context of Matrix grammars is a lexical entry or lexical rule instance that has received all the inflection it needs to serve as the daughter of a phrase structure rule. This status is indicated by the feature INFLECTED. This value can change between the daughter and the mother, or remain the same. Lexeme-to-word rules are identified by being the outermost non-optional rule. Pseudocode for the function FIND_RULETYPE() appears in (15). As each affix is being processed, I search the choices file for rules that this rule can serve as the daughter to. If it can be the daughter of another

[1] See Appendix C for a tree of the full rule hierarchy for Zulu

```
(15)  1  function FIND_RULETYPE(morph, basetypes):
             if CHOICES.GET_OPT(morph) == 'no' and
                IS_LAST_NONOPT(morph) == yes:
                return 'lexeme-to-word'
             else if CHOICES.GET_OPT(morph) == 'yes' or
                IS_LAST_NONOPT(morph) == yes:
         5      return 'word-to-lexeme'
             else if IS_FORCED(morph):
                return 'lexeme-to-word'
             else:
                return 'lexeme-to-lexeme'
```

non-optional rule, then it is not a lexeme to word rule. If it is non-optional but it can be the daughter of an optional rule, I recursively search the choices files to see if there is another non-optional rule that can occur after this one. (This occurs in the function IS_LAST_NONOPT()). If this rule is non-optional and there are no non-optional rules that could follow this one or if this rule cannot be the daughter of another rule, then it is a lexeme-to-word rule (lines 2-3). It may also be a lexeme-to-word rule if this rule is listed as the `forces` value of another morpheme in the choices file, if that morpheme is a word-to-lexeme rule (lines 6-7). The rule is a word-to-lexeme rule if it cannot be followed by a non-optional rule, but it has designated a `forces` value in the choices file (lines 4-5). If none of the conditions described in this paragraph apply, the rule is a lexeme-to-lexeme rule (lines 8-9). Because this is an if-else structure, one of these options is always selected.

4.2.4 Inflecting vs. Constant Rules

The other piece of information needed to create a new rule is whether this rule is an inflecting rule or a constant rule. Inflecting rules add or modify orthographic material, while constant rules leave the orthography unchanged between input and output. The inflecting vs

constant distinction introduces a different sort of complication than determining the daughter value or if this is a ltow, wtol, or ltow, because it involves knowledge about the paradigm itself. As mentioned above, some paradigms have a single possible value, while others have a range of values. The way this is specified in the grammar is to have a parent rule type for the paradigm as a whole, and the individual values inherit from that type. So if there is an inflectional paradigm that specifies tense, there would be a supertype type containing all the rule information common to all the tense rules, and then individual subtype rules for the specific paradigm values, e.g. past or non-past. If this paradigm has rule subtypes, it is these subtypes that would specify the orthographic form (or lack thereof) rather than the supertype. Since I want to push as much information onto the supertype as possible, the system determines whether the subtypes are all inflectional rules, or a mix of constant and inflectional rules.[2] If the subtypes are all inflectional, the supertype can be defined as an inflectional rule type. Note that this does not make the supertype itself a valid inflectional rule, and it will not have an associated spelling-change rule instance.

In CREATE_BASIC_RULE_TYPE() (16), the sole rule or supertype rule is created. If this paradigm has an orth value of 'NONE' either on the top-level paradigm or in one of the subtypes, there has to be a constant-lex-rule (HAS_CONSTANT_RULE() (line 2)). If there are subtype rules (line 3), then there must also be inflectional subtype rules, because I am assuming there is not a paradigm containing multiple constant rules. The parent type therefore can't specify whether the rule is inflecting or constant, and so in line 4, the first rule definition is created, with the rule type inheriting from the ltow, wtol, or ltol type, as was determined by FIND_RULETYPE(). If there are no subtypes, then the rule definition

[2] I assume that a paradigm with multiple values would not contain only constant rules. Such a grammar could be constructed in this framework, but each rule subtype would be individually designated as constant, rather than pushing this constraint onto the supertype.

(16)
```
1  function CREATE_BASIC_RULE_TYPE(morph, basetypes, daughtertype,
                                   ruletype, rulename):
       if HAS_CONSTANT_RULE(CHOICES.GET_PARADIGM(morph)):
          if CHOICES.GET_ORTH(CHOICES.GET_PARADIGM(morph)) != 'NONE':
             GRAMMAR.ADD(rulename + ':=' +ruletype)
5         else:
             GRAMMAR.ADD(rulename+ ':=' + ruletype + '& constant-lex-rule')
             LRULES.ADD(rulename)
       else:
          GRAMMAR.ADD(rulename + ':=' + ruletype + '& inflecting-lex-rule')
10     GRAMMAR.ADD(rulename + ':= [DTR '+ daughtertype + ']')
       if ruletype == 'lexeme-to-word':
          for basetype in basetypes:
             GRAMMAR.ADD(basetype + ':= [INFLECTED -]')
       if LENGTH.(basetypes) == 1 and CHOICES.NUM_FORCES(basetype[0]) == 1:
15        GRAMMAR.ADD(rulename + ':= [INFLECTED -]')
```

can also inherit from constant-lex-rule (line 6).[3] A rule instance inheriting from this type is added to the list of constant lexical rules (line 7). If there are no constant rules, then the whole paradigm must be inflecting. At this point we can have this lexical rule inherit from `inflecting-lex-rule` (lines 8-9). In the Zulu tense example, the rule at this point is:

(17)
```
tense-lex-rule := lexeme-to-lexeme-rule.
```

Once the basic rule definition is created, we can add things to it. There customization system includes helper functions for adding new rules and extending rule definitions. After the initial rule name and inheritance is defined, we can add some further constraints on the feature structure, including the daughter value as defined earlier. So, in the Zulu tense example, the rule definition ends up as:

[3] These types are cross-classified in the actual grammar, making e.g. `constant-ltow-rule`, but I did not include this in the pseudocode

(18)
```
tense-lex-rule := lexeme-to-lexeme-rule &
    [ DTR tense-rule-dtr ].
```

(19)
```
1  function CREATE_RULE_SUBTYPES(pdm, rulename):
       if CHOICES.HAS_MORE_AFFS(pdm):
          whileCHOICES.HAS_MORE_AFFS(pdm):
             aff ← CHOICES.GET_NEXT_AFF(pdm)
5            affrulename ← CHOICES.GET_NAME(aff) + '-lex-rule'
             if CHOICES.GET_ORTH(aff) == 'NONE':
                GRAMMAR.ADD(affrulename + ':=' + rulename +
                                  '& constant-lex-rule')
                LRULES.ADD(affrulename)
10           else:
                if HAS_CONSTANT_RULE(pdm):
                   GRAMMAR.ADD(affrulename + ':=' + rulename +
                                     '& inflecting-lex-rule')
                else:
15                 GRAMMAR.ADD(rulename + ':=' + rulename)
                IRULES.ADD(affrulename, CHOICES.GET_ORTH(aff))
       else:
          if CHOICES.GET_ORTH(pdm) != 'NONE':
             IRULES.ADD(rulename, CHOICES.GET_ORTH(pdm))
```

In CREATE_RULE_SUBTYPES() (19), this process is essentially repeated for each of the subtype affixes, if any. After the creation of each rule, it is added to the appropriate list of rule instances, as discussed in §2.1.2.

4.2.5 Tracking co-occurrence constraints

If the choices file indicates that this rule must or cannot co-occur with another rule, I need a way to check if the required (or disallowed) rule has already applied. Rules can only see the rule that previously applied, as the type of their daughter value. To solve this problem, I create the complex feature TRACK, and add it to the lexical rule type definitions

if necessary for the language. The TRACK feature, if required, is added as a top-level feature to the type word-or-lexrule. Each rule that is added gets a feature within the track feature that is named after the rule name.

(20)
```
1  function CREATE_TRACK_FEATURES(morph, basetypes):
       tracks ← []
       while CHOICES.HAS_MORE_FORCES(morph):
         forces ← CHOICES.GET_FORCES(morph)
5        trackfeature ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(forces))
         for basetype in basetypes:
           GRAMMAR.ADD(basetype + ':= [TRACK.f-' + trackfeature+ ' −]')
       while CHOICES.HAS_MORE_REQS(morph):
         reqs ← CHOICES.GET_REQS(morph)
10       trackfeature ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(reqs))
         for basetype in basetypes:
           GRAMMAR.ADD(basetype + ':= [TRACK.r-' + trackfeature+ ' −]')
         tracks.APPEND(morph)
       while CHOICES.HAS_MORE_DISREQS(morph):
15       disreqs ← CHOICES.GET_DISREQS(morph)
         trackfeature ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(disreqs))
         for basetype in basetypes:
           GRAMMAR.ADD(basetype + ':= [TRACK.d-' + trackfeature+ ' +]')
         tracks.APPEND(morph)
20     return tracks
```

As I iterate over each morpheme, I call CREATE_TRACK_FEATURES() (20) to see if this morpheme has any `req`, `disreq`, or `forces` values specified in the choices file. If so, I add the TRACK feature to the definition of the lexical types that this rule ultimately attaches to (lines 6-7, 11-12, 17-18). The value of TRACK is itself a bundle of features: the features are the names of lexical rules, and the values are boolean values indicating whether that rule is valid to apply or not.

After iterating over all the morphemes, I then add the TRACK feature to the lexical

(21)
```
1  function ADD_SINGLE_TRACKS(tracks):
     for track in tracks:
       mother ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(track))
       while CHOICES.HAS_MORE_MORPHS():
5        morph ← CHOICES.GET_NEXT_MORPH()
         daughter ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(morph))
         if CHOICES.FORCED_BY(track, morph):
           GRAMMAR.ADD(mother + '-lex-rule := [TRACK.f-' + mother + '−]')
           GRAMMAR.ADD(daughter + '-lex-rule := [TRACK.f-' + mother + '+]')
10         GRAMMAR.ADD(daughter + '-lex-rule := [DTR.TRACK.f-' + mother + ' −]')
         else if morph in tracks:
           GRAMMAR.ADD(mother + '-lex-rule := [TRACK.f-' + mother + ' #]')
           GRAMMAR.ADD(daughter + '-lex-rule := [TRACK.f-' + mother + ' #]')
         if CHOICES.REQUIRED_BY(track, morph):
15         GRAMMAR.ADD(mother + '-lex-rule := [TRACK.r-' + mother + '−]')
           GRAMMAR.ADD(mother + '-lex-rule := [DTR.TRACK.r-' + mother + ' +]')
           GRAMMAR.ADD(daughter + '-lex-rule := [TRACK.r-' + mother + '+]')
           GRAMMAR.ADD(daughter + '-lex-rule := [DTR.TRACK.r-' + mother + '−]')
         else if morph in tracks:
20         GRAMMAR.ADD(mother + '-lex-rule := [TRACK.r-' + mother + ' #]')
           GRAMMAR.ADD(daughter + '-lex-rule := [TRACK.r-' + mother + ' #]')
         if CHOICES.DISALLOWED_BY(track, morph):
           GRAMMAR.ADD(mother + '-lex-rule := [TRACK.d-' + mother + '−]')
           GRAMMAR.ADD(mother + '-lex-rule := [DTR.TRACK.d-' + mother + '+]')
25         GRAMMAR.ADD(daughter + '-lex-rule := [TRACK.d-' + mother + '−]')
           GRAMMAR.ADD(daughter + '-lex-rule := [DTR.TRACK.d-' + mother + '+]')
         else if morph in tracks:
           GRAMMAR.ADD(mother + '-lex-rule := [TRACK.d-' + mother + ' #]')
           GRAMMAR.ADD(mother + '-lex-rule := [DTR.TRACK.d-' + mother + ' #]')
```

rules in ADD_SINGLE_TRACKS() (21). Every rule that has a co-occurance restriction listed in the choices file constrains its DTR to have the track feature corresponding to itself (that is, the mother) to be positive, indicating that this rule can apply (lines 8, 15, 23). Every rule that is required or disallowed by another rule toggles the TRACK feature corresponding to the rule placing the constraint, indicating that the requirement has been fulfilled or that

(22)
```
1  function COPY_ALL_TRACKS(tracks)
       while CHOICES.HAS_MORE_MORPHS():
         morph ← CHOICES.GET_NEXT_MORPH()
         if morph not in tracks:
5            rulename ← CHOICES.GET_NAME(CHOICES.GET_PARADIGM(morph))
                 GRAMMAR.ADD(rulename + '-lex-rule := [TRACK '#]')
                 GRAMMAR.ADD(rulename + '-lex-rule := [DTR.TRACK ' #]')
```

the rule can no longer apply (lines 9-10, 16-18, 24-26). Then, if there are other features being tracked that aren't relevant to the current rule, all those features need to be explicitly copied between the daughter and the mother, to prevent that information from being lost (lines 11-13, 19-21, 27-29). Rules are not involved in co-occurrance restrictions (neither placing requirements or being required) copy the entire TRACK feature up unmodified in COPY_ALL_TRACKS() (22).

Here's a sample rule from Slave. This is the incorporated postposition rule, which is a word-to-lexeme rule. It is tracking the feature TRACK.POSTPOS-OBJECT. The other TRACK feature number is unchanged and so copied up between the mother and the daughter:

```
incorp-postpos-lex-rule := word-to-lexeme-rule &
                           inflecting-lex-rule &
                           postpos-object-lex-rule-dtr &
                           negation-rule-dtr &
  [ TRACK [ POSTPOS-OBJECT +,
                    NUMBER #track ] ,
    DTR incorp-postpos-rule-dtr &
            [ TRACK [ POSTPOS-OBJECT -,
                    NUMBER #track ] ] ].
```

The value of DTR.TRACK.POSTPOS-OBJECT is set to − because the postpositional

object affix could not apply to the daughter. The postpositional object affix requires that the incorporated postposition is applied before it.[4] On the rule output, the POSTPOS-OBJECT feature is set to +, indicating that it is now able to be applied.

```
postpos-object-lex-rule := word-to-lexeme-rule &
                           inflecting-lex-rule &
                           negation-rule-dtr &
 [ TRACK [ POSTPOS-OBJECT -,
           NUMBER #track ] ,
   DTR incorp-postpos-rule-dtr &
           [ TRACK [ POSTPOS-OBJECT +,
                     NUMBER #track ] ] ].
```

Similarly, the postpos-object lexical rule requires that the input is DTR.TRACK.POSTPOS-OBJECT be set to +, meaning that it can only occur if the incorporated postposition has been applied. The output sets the feature value back to − to indicate that the requirement has been met.

4.2.6 Summary

The customization script does the work of creating lexical rules based on the choices file. I presented here an overview of the algorithm that builds the lexical rules. The design goals of ordering lexical rules, identifying lexeme vs word status, and tracking co-occurance restrictions have all been implemented. In addition, this implementation can identify constant vs inflecting rules and apply this information at the correct level in the rule hierarchy, and create intermediate rule types where necessary to aid in rule ordering.

---

[4] These morphemes are prefixes being applied right to left, so the resulting word will have the postposition following the postpositional object. See example (6) in §3.4.1

4.3 Summary

In this section, I have outlined the implementation of the morphotactic customization system. I have implemented a new range of values in the choices file, specifying paradigm information and meta-paradigm information needed to define a system of lexical rule types. I then updated our customization script to process the new choices file, creating a skeleton structure of lexical rules and type hierarchy.

Chapter 5

EVALUATION

The goal of this work is to create a system that will work for any natural language. With this goal in mind, the system is developed by iteratively creating test suites for a series of typologically and genetically diverse languages, and modifying the choices file specifications and the customization script to cover any phenomena necessary to accurately parse all the languages covered up to that point. I evaluate the system by measuring the system performance on each test suite, as well as the amount of work necessary to bring the new system up to 100% on each successive language. The languages used for testing were Zulu, Slave, Finnish, and Uzbek. These languages are typologically diverse, and come from different language families (Niger-Congo, Na-Dene, Uralic and Altaic, respectively) but each is morphologically complex. I was able to obtain 100% coverage across these language for the phenomena I was addressing. In fact, after reaching 100% on the first three languages, no additional modification was necessary to get 100% on the Uzbek test suite.

5.1 Languages Used

5.1.1 Zulu

Zulu is Bantu language mainly spoken in South Africa. Bantu languages are distinctive for having a large variety of noun classes. These classes are equivalent to grammatical gender, but rather than two or three genders as is typical, Zulu has about 15 noun classes. In my Zulu grammar, nouns inflect to show their noun class, and verbs can inflect to show tense,

negation, and agreement with the noun class of the subject and object. There is also final suffix that can take various values that contribute some syntactic features, but also has a default value if no special cases apply. The Zulu morphological information was derived from Nyembezi and Doke 1979 and Poulos and Bosch 1997.

5.1.2 Slave

Slave is an Athabaskan language spoken in western Canada. This language was selected precisely because of its wide array of inflectional morphology. There are 16 affix types that can attach to a verb, with some affix types able to appear more than once. While I do not claim a native speaker would accept this particular sample, here is one test suite item I constructed, based on Rice's (1989) descriptive grammar:

(23) du-be-keh-na-ya-dlo-leh-ele-ne-i-w-n-id-d-tah
NEG-3SG-into-THM-DISTR-laugh-DU-RECP-THM-SER-wCONJ-PFV-1DU-RECP-move.foot
'We two didn't kick each other into it (e.g. a hole) repeatedly while laughing.'

5.1.3 Finnish

Finnish is a Uralic language spoken mainly in Finland. While Finnish does have extensive derivational morphology, I chose here to focus on the inflectional morphology. Nouns are marked for number, case, possession, and can also take a variety of particles that mark questions or emphasis. Verbal inflection consists of person/number agreement with the subject, tense and mood markers and a passive or indefinite marker. Verbs can also take a number of the same particles that apply to the nouns. I based this test suite on the description in Karlsson 1983.

5.1.4 Uzbek

Uzbek is an Altaic language spoken mainly in Uzbekistan. The test suite I created was guided by the description in Sjoberg 1963. The inflectional morphology I concentrated on was plural marking, possession, and case for the nouns, and negation, two levels of tense/aspect/modality (TAM) marking, person and number agreement, and question marking for verbs. Uzbek has a complex system of agreement between the TAM markers and the surface forms of the person/number markers. There are 4 series of subject markers that vary in form based on which TAM marker is used. For example:

(24) *kel-gan-miz*
come-PST.PRF-1PL
'We came, we have come.'

(25) *kel-di-k*
come-PRET-1PL
'We came.'

In (13), the TAM marker is past perfect, while in (14) it is preterite. Each of these tenses chooses from a different set of person/number markers. This provides a good example of the differentiation between morphophonology and morphosyntax. The variation is in the surface form only; while it would be possible to model this distinction in the syntax (i.e. have the verb choose the person marker based on the TAM marker), there is not a meaning difference between the person/number markers in the two examples. For present purposes, I chose to handle them on the string level, working with morpheme glosses rather than creating rules pairing sets of person+number markers with particular combinations of tense/aspect/modality markers. This allows the constructed words to be linguistically

accurate without building excessive complication into the grammar. The differentiation could still be made later, once the rules have access to the necessary syntactic information.

5.2 Test Suite Design

For each language, I designed a test suite of grammatical and ungrammatical items to illustrate the interesting morphotactic phenomena for that language. For Zulu, Finnish, and Uzbek, test suites had previously been developed in conjunction with more extensive grammars.[1] These test suites were more extensive both in phenomena covered and vocabulary used than the testsuites required for our purposes, but they were useful as a starting point and reference. The Slave test suite was built from scratch. All the test suites have been regularized for morphophonology, in some cases going so far as to use morpheme glosses.

Vocabulary was limited to that which could be input on the customization page, as I am testing the abilities of customization rather than Matrix grammars in general. Vocabulary consisted of two nouns, a transitive verb, and an intransitive verb. When an inflectional paradigm had more than two or three possible values, I used only a subset of these to illustrate the morphotactic phenomena.

Specific test items were created by starting with the shortest well-formed string (that is, one without any optional morphemes) and permuting the order of the affixes to create ungrammatical examples. Optional morphemes were then added individually, and ungrammatical examples created by shifting the new affix to each position in the string, rather than creating every new permutation, as this would have increased the test suite size exponentially as new morphemes were added. The last item in the test suite was the longest grammatical string that could be constructed, using as many affixes as possible.

I previously listed many phenomena this system was not designed to cover, includ-

[1]The Zulu grammar was developed by me, the Finnish grammar by Ryan Mattson, and the Uzbek grammar by Michael Tepper in the context of LING 567 at the University of Washington (Bender 2007)

ing effects of morphology on phrase-level syntax, and any restrictions that require some knowledge about the meaning of the particular affixes. These phenomena were intentionally excluded from the test suites, because correctly parsing these sentences was not the intention of the current project.

Test suites were run and their performance tracked using [incr tsdb()] test environment (Oepen 2001).

5.3 Development And Evaluation Process

The first language used as a reference for development was Zulu. A choices file was written by hand, and the customization script expanded to process the choices file. As I added/changed code in the customization script, I would build a grammar using the choices file, and use the resulting grammar to run the test suite. I continued this process until the grammar could parse all and only the grammatical strings in the test suite.

Once the customization system produced a grammar for Zulu, I built a test suite and choices file for Slave. I created a Slave grammar using the "version Zulu" system, that is, the customization script that was sufficient to create an adequate Zulu grammar. The Slave grammar on the version Zulu system was not able to get 100% on the test suite, so I revised the customization script, and in this case the choices file specifications, until the Slave grammar was at 100% on the test suite. I then repeated this process for Finnish and Uzbek. While additional modifications were required to bring Finnish up to 100% on the test suite, Uzbek did not expose any additional oversights in either the choices file format or the customization script.

Each test suite was developed at the time I began work on that language version. While this did allow me to create each test suite with the knowledge acquired from the previous versions in mind, it did not allow me to run the version in development over the test suites

	v.Zulu	v.Slave	v.Finnish
Zulu	100.0/0.0	100.0/0.0	100.0/0.0
Slave	76.5/9.6	100.0/0.0	100.0/0.0
Finnish	42.9/9.1	*/*	100.0/0.0
Uzbek	25.0/0.0	100.0/0.0	100.0/0.0

Table 5.1: Test suite performance on each version of the choices file and customization script, given by percent coverage / percent overgeneration. Version Slave customization was unable to produce a valid grammar for Finnish, and was therefore untestable.

for languages I had not tested for yet. However, I did run versions in development over the test suites for previous languages as a form of regression testing. For example, at the start of development for Version Slave, the system got 100% on the Zulu test suite. After I reached 100% on the Slave test suite, I made sure I was still getting 100% on the Zulu test suite before starting development on Version Finnish.

As each version was determined complete, the system was frozen in that form. This was done by tagging that revision in our Subversion repository. After development for all versions was complete, I took each frozen version and used it to build grammars for all the languages, and ran the test suites. I recorded the percentage coverage and overgeneration as:

(26) coverage = (number of grammatical test suite items parsed / total grammatical items)

(27) overgeneration = (number of ungrammatical test items parsed / total ungrammatical items).

These results are shown in table 5.1.

I also wanted to measure how much work was required to bring a new language up to 100%. While not necessarily a good measure of how much time or mental energy was

no inflection to Zulu	7%
Zulu to Slave	10%
Slave to Finnish	2%
Finnish to Uzbek	0%

Table 5.2: Percent change in the customization script to get 100% test suite accuracy, calculated as (number of lines added + number of lines removed) / number of lines in starting version.

required, I chose to look at how much the customization script changed between versions. I ran a diff between consecutive pairs of versions and calculated the percent change as :

(28) (number of lines added + number of lines removed) / total number of lines in the script before the change

So for version Zulu to version Slave, I took:

(29) (number of lines added in v.Slave + number of lines removed from v.Zulu) / total number of lines in v.Zulu

These results are shown in table 5.2.

5.4 Results

Version Zulu was “blind” to the other languages, as no feedback from the performance of other languages went into its development. Similarly, Version Slave was “blind” to Uzbek and Finnish, as was Version Finnish to Uzbek. It is therefore unsurprising that the other languages did not do as well in Version Zulu. The codebase changed more between version Zulu and version Slave than between the pre-inflection system and version Zulu. I expected the other three languages to perform equal or better in version Slave than in version Zulu. This is in fact the case for Zulu and Uzbek (and trivially Slave), but version Slave was

unable to produce a valid grammar for Finnish at all, and so was untestable. This was due to a minor oversight,[2] and so with relatively few changes to the customization script, I was able to get Finnish working. Version Finnish is able to produce adequate grammars for all four languages. No modifications were necessary to get 100% on the Uzbek test suite, so no additional version was created.

I find these results promising. While there are certainly more morphotactic phenomena that need to be developed and tested, I was able to reach a point of convergence for four disparate languages. This suggests that the current system would work for a much larger number of the world's languages.

[2]Version Slave did not allow for an affix to attach to multiple lexical types, a phenomenon which occurs in Finnish. This produced a python error when running the customization script. It was easily corrected by storing the lexical types as a list rather than a single variable

Chapter 6

CONCLUSION

The goal of the Grammar Matrix is to provide a cross-linguistic resource for creating rule-based precision grammars. The customization system is meant to provide a fast and straightforward way to create starter grammars to jump-start the development process. The more structure and functionality provided in the customization system, the more complete the starter-grammars will be. Languages utilize inflectional morphology for a wide range of syntactic and semantic functions. By providing a general morphotactic framework, I provide a general resource for future development of the customization system. As more functionality is added, the relevant feature structures can be added to the lexical rules. A common method for creating lexical rules provides consistency in rule naming, typing, and structure across the grammar and ensures that lexical rules added for different grammatical phenomena interact properly with each other.

BIBLIOGRAPHY

Beesley, Kenneth R., and Lauri Karttunen. 2003. *Finite State Morphology*. Stanford CA: CSLI Publications.

Bender, Emily M. 2007. Combining research and pedagogy in the development of a crosslinguistic grammar resource. In T. H. King and E. M. Bender (Eds.), *Proceedings of the GEAF07 Workshop*, 26–45, Stanford, CA. CSLI.

Bender, Emily M., and Dan Flickinger. 2005. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos)*, Jeju Island, Korea.

Bender, Emily M., Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of the Workshop in Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, 8–14, Taipei, Taiwan.

Bender, Emily M., and Jeff Good. 2005. Implementation for discovery: A bipartite lexicon to support morphological and syntactic analysis. In *Proceedings of the 41st Annual Meeting of the Chicago Linguistic Society*.

Bender, Emily M., Laurie Poulson, Scott Drellishak, and Chris Evans. 2007. Validation and regression testing for a cross-linguistic grammar resource. In *ACL 2007 Workshop on Deep Linguistic Processing*, 136–143, Prague, Czech Republic. Association for Computational Linguistics.

Booij, G.E. 2005. *The grammar of words: an introduction to linguistic morphology*. Oxford: Oxford University press.

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The parallel grammar project. In J. Carroll, N. Oostdijk, and R. Sutcliffe (Eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, 1–7.

Callmeier, Ulrich. 2000. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6 (1) (Special Issue on Efficient Processing with HPSG):99 – 108.

Comrie, Bernard. 1981. *Language universals and linguistic typology: syntax and morphology*. Chicago: University of Chicago Press.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. Stanford, CA: CSLI Publications.

Drellishak, Scott, and Emily M. Bender. 2005. A coordination module for a crosslinguistic grammar resource. In S. Müller (Ed.), *The Proceedings of the 12th International Conference on Head-Driven Phrase Structure Grammar, Department of Informatics, University of Lisbon*, 108–128, Stanford. CSLI Publications.

Goldsmith, John. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics* 27(2):153–198.

Karlsson, Fred. 1983. *Finnish Grammar*. Porvoo, Finland: WSOY.

Kim, Roger, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, Hiroshi Masuichi, and Tomoko Ohkuma. 2003. Multlingual grammar development via grammar porting. In E. M. Bender, D. Flickinger, F. Fouvry, and M. Siegel (Eds.), *Proceedings of the ESSLLI 2003 Workshop "Ideas and Strategies for Multilingual Grammar Development"*, 49–56, Vienna, Austria.

Koskenniemi, Kimmo. 1984. A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics*.

Krieger, Hans-Ulrich, and Ulrich Schafer. 1994. Tdl – a type description language for constraint-based grammars. In *Proceedings of the 15th international Conference on Computational Linguistics*, 893–899, Kyoto, Japan.

Maxwell, Mike. 2002. Resources for morphology learning and evaluation. In *LREC 2002: Third International Conference on Language Resources and Evaluation*, Vol. III, 967–974.

Maxwell, Mike, Gary Simons, and Larry Hayashi. 2002. A morphological glossing assistant. In *Proceedings of the International LREC Workshop on Resources and Tools in Field Linguistics*.

McShane, Marjorie, and Sergei Nirenberg. 2003. Blasting open a choice space: Learning inflectional morphology for NLP. *Computational Intelligence* 19(2):111–135.

McShane, Marjorie, Sergei Nirenburg, Jim Cowie, and Ron Zacharski. 2002. Embedding knowledge elicitation and mt systems within a single architecture. *Machine Translation* 17(4):271–305.

Nyembezi, CL Sibusiso, and Clement Martyn Doke. 1979. *Learn Zulu*. Pietermaritzburg: Shuter and Shooter.

Oepen, Stephan. 2001. [incr tsdb()] — Competence and performance laboratory. User manual. Technical report, COLI, Saarbrücken, Germany.

Oepen, Stephan, Erik Velldal, Jan Tore Lnning, Paul Meurer, Victoria Rosn, and Dan Flickinger. 2007. Towards hybrid quality-oriented machine translation. On linguistics and probabilities in MT. In *TMI:07*, Skvde, Sweden.

Pollard, Carl, and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL and Stanford, CA: The University of Chicago Press and CSLI Publications.

Poulos, G, and Sonja E Bosch. 1997. *Zulu*. Munich: LINCOM Europa.

Rice, Keren. 1989. *A grammar of Slave*. Vol. 5 of *Mouton grammar library*. Berlin: Mouton de Gruyter.

Salminen, Tapani. 1997. *Tundra Nenets inflection*. Helsinki: Suomalais-ugrilainen seura.

Sjoberg, Andreè F. 1963. *Uzbek Structural Grammar*. Vol. 18 of *Uralic and Altaic Series*. Bloomington: Indiana University.

Appendix A

CHOICES FILES

These are the choices files from which the test grammars were created. Please note that these were designed for modeling and testing a particular set of morphological phenomena, and not as true starter grammars for the languages in question. For example most of the morpheme paradigms have only one or two affixes listed, where in the actual language there are many more. They also contain a hodgepodge of orthographic forms and morpheme glosses, largely depending on which was the simplest for me to encode. Some decisions I made regarding morpheme ordering will probably be different than the analysis of a linguist more familiar with the language. In addition, I did not answer parts of the questionnaire not directly relevant to my work. I am including these documents here simply as examples of the choices file format.

A.1 Zulu

```
version=2
section=language
language=Zulu

section=word-order
word-order=svo
has-dets=no

section=sentential-negation
infl-neg=on
neg-infl-type=main
neg-aff=prefix
neg-aff-orth=NEG-
```

```
section=coordination

section=matrix-yes-no
q-part=on
q-part-order=after
q-part-orth=na

section=inflection
morph1_type=pdm1
morph1_aff=prefix
morph1_opt=no
morph1_dtr1_type=noun
morph2_type=pdm3
morph2_aff=prefix
morph2_opt=yes
morph2_dtr1_type=tverb
morph3_type=pdm4
morph3_aff=prefix
morph3_opt=yes
morph3_dtr1_type=iverb
morph3_dtr2_type=morph2
morph4_type=pdm2
morph4_aff=prefix
morph4_opt=no
morph4_dtr1_type=morph3
morph5_type=pdm5
morph5_aff=suffix
morph5_opt=no
morph5_dtr1_type=morph4
morph5_dtr2_type=morph6
morph6_type=pdm6
morph6_aff=prefix
morph6_opt=yes
morph6_dtr1_type=morph4

section=infl-paradigms
pdm1_name=NC
pdm1_aff1_name=C1
pdm1_aff1_orth=C1-
pdm1_aff2_name=C9
pdm1_aff2_orth=C9-
```

pdm2_name=SC
pdm2_aff1_name=SC1
pdm2_aff1_orth=SC1-
pdm2_aff2_name=SC9
pdm2_aff2_orth=SC9-
pdm3_name=OC
pdm3_aff1_name=OC1
pdm3_aff1_orth=OC1-
pdm3_aff2_name=OC9
pdm3_aff2_orth=OC9-
pdm4_name=tense
pdm4_aff1_name=fut-tense
pdm4_aff1_orth=FUT-
pdm4_aff2_name=pres-tense
pdm4_aff2_orth=PRES-
pdm5_name=FV
pdm5_aff1_name=default-FV
pdm5_aff1_orth=-a
pdm5_aff2_name=neg-FV
pdm5_aff2_orth=-e
pdm6_name=negation
pdm6_orth=NEG-

section=basic-lexicon
noun1=ntwana
noun1_pred=_child_n_rel
noun1_det=imp
noun2=yoka
noun2_pred=_snake_n_rel
noun2_det=imp
iverb=dlal
iverb-pred=_play_v_rel
iverb-subj=np
tverb=bon
tverb-pred=_see_v_rel
tverb-subj=np
tverb-obj=np

section=test-sentences
sentence1=C1-ntwana SC1-FUT-dlal-a
sentence2=C1-ntwana SC1-FUT-bon-a C9-yoka

A.2 Slave

version=2

section=language
language=Slave

section=word-order
word-order=sov
has-dets=no

section=sentential-negation

section=coordination

section=matrix-yes-no

section=inflection
morph1_type=pdm1
morph1_aff=prefix
morph1_opt=yes
morph1_dtr1_type=verb
morph2_type=pdm2
morph2_aff=prefix
morph2_opt=no
morph2_dtr1_type=morph1
morph3_type=pdm3
morph3_aff=prefix
morph3_opt=no
morph3_dtr1_type=morph2
morph4_type=pdm4
morph4_aff=prefix
morph4_opt=yes
morph4_dtr1_type=morph3
morph5_type=pdm5
morph5_aff=prefix
morph5_opt=yes
morph5_dtr1_type=morph4
morph6_type=pdm6

morph6_aff=prefix
morph6_opt=yes
morph6_dtr1_type=morph5
morph7_type=pdm7
morph7_aff=prefix
morph7_opt=yes
morph7_dtr1_type=morph6
morph8_type=pdm8
morph8_aff=prefix
morph8_opt=yes
morph8_dtr1_type=morph7
morph9_type=pdm9
morph9_aff=prefix
morph9_opt=yes
morph9_dtr1_type=morph8
morph9_disreq1_type=morph7
morph10_type=pdm10
morph10_aff=prefix
morph10_opt=yes
morph10_dtr1_type=morph9
morph11_type=pdm11
morph11_aff=prefix
morph11_opt=yes
morph11_dtr1_type=morph10
morph12_type=pdm12
morph12_aff=prefix
morph12_opt=yes
morph12_dtr1_type=morph10
morph13_type=pdm13
morph13_aff=prefix
morph13_opt=yes
morph13_dtr1_type=morph11
morph13_dtr2_type=morph12
morph14_type=pdm14
morph14_aff=prefix
morph14_opt=yes
morph14_dtr1_type=morph13
morph14_forces1_type=morph15
morph15_type=pdm15
morph15_aff=prefix
morph15_opt=yes

morph15_dtr1_type=morph14
morph15_req1_type=morph14
morph16_type=pdm16
morph16_aff=prefix
morph16_opt=yes
morph16_dtr1_type=morph15

section=infl-paradigms
pdm1_name=classifier
pdm1_aff1_name=reciprocal
pdm1_aff1_orth=d-
pdm2_name=subject
pdm2_aff1_name=1SG-SUBJ
pdm2_aff1_orth=h-
pdm2_aff2_name=1PL-SUBJ
pdm2_aff2_orth=id-
pdm2_aff3_name=3-SUBJ
pdm2_aff3_orth=NONE
pdm3_name=mode
pdm3_aff1_name=perfective
pdm3_aff1_orth=n-
pdm3_aff2_name=imperfective
pdm3_aff2_orth=NONE
pdm4_name=conjugation
pdm4_aff1_name=yconj
pdm4_aff1_orth=y-
pdm4_aff2_name=wconj
pdm4_aff2_orth=w-
pdm5_name=aspect
pdm5_aff1_name=inceptive
pdm5_aff1_orth=de-
pdm5_aff2_name=serative
pdm5_aff2_orth=i-
pdm6_name=theme
pdm6_aff1_name=theme1
pdm6_aff1_orth=ne-
pdm7_name=deixis
pdm7_aff1_name=3PL
pdm7_aff1_orth=ke-
pdm8_name=object
pdm8_aff1_name=3SG

pdm8_aff1_orth=be-
pdm8_aff2_name=1SG-OBJ
pdm8_aff2_orth=se-
pdm8_aff3_name=reciprocal-object
pdm8_aff3_orth=ele-
pdm9_name=number
pdm9_aff1_name=dual
pdm9_aff1_orth=leh-
pdm10_name=incorporated-stem
pdm10_aff1_name=laugh
pdm10_aff1_orth=dlo-
pdm11_name=customary
pdm11_orth=na-
pdm12_name=distributive
pdm12_orth=ya-
pdm13_name=adverbial
pdm13_aff1_name=avbl1
pdm13_aff1_orth=na-
pdm14_name=incorporated-postposition
pdm14_aff1_name=into
pdm14_aff1_orth=keh-
pdm15_name=postpos-object
pdm15_aff1_name=3-POSTPOS-OBJ
pdm15_aff1_orth=be-
pdm16_name=negation
pdm16_orth=du-

section=basic-lexicon
noun1=teere
noun1_pred=_girl_n_rel
noun1_det=imp
noun2=soba
noun2_pred=_money_n_rel
noun2_det=imp
iverb=d-shin
iverb-pred=_sing_v_rel
iverb-subj=np
tverb=tah
tverb-pred=_kick_v_rel
tverb-subj=np
tverb-obj=np

```
section=test-sentences
sentence1=teere d-shin
sentence2=h-d-shin
```

A.3 Finnish

```
version=2

section=language
language=Finnish

section=word-order
word-order=svo
has-dets=no

section=sentential-negation

section=coordination

section=matrix-yes-no

section=basic-lexicon
noun1=opiskelija
noun1_pred=_student_n_rel
noun1_det=imp
noun2=omena
noun2_pred=_apple_n_rel
noun2_det=imp
iverb=kAvele
iverb-pred=_walk_v_rel
iverb-subj=np
tverb=pidA
tverb-pred=_like_v_rel
tverb-subj=np
tverb-obj=np

section=inflection
morph1_type=pdm1
morph1_aff=suffix
```

morph1_opt=no
morph1_dtr1_type=noun
morph2_type=pdm2
morph2_aff=suffix
morph2_opt=no
morph2_dtr1_type=morph1
morph3_type=pdm3
morph3_aff=suffix
morph3_opt=yes
morph3_dtr1_type=morph2
morph4_type=pdm4
morph4_aff=suffix
morph4_opt=yes
morph4_dtr1_type=morph3
morph4_dtr2_type=morph8
morph5_type=pdm8
morph5_aff=suffix
morph5_opt=yes
morph5_dtr1_type=verb
morph6_type=pdm6
morph6_aff=suffix
morph6_opt=yes
morph6_dtr1_type=morph5
morph7_type=pdm7
morph7_aff=suffix
morph7_opt=yes
morph7_dtr1_type=morph5
morph8_type=pdm5
morph8_aff=suffix
morph8_opt=no
morph8_dtr1_type=morph6
morph8_dtr2_type=morph7

section=infl-paradigms
pdm1_name=number
pdm1_aff1_name=plural
pdm1_aff1_orth=-PL
pdm1_aff2_name=singular
pdm1_aff2_orth=NONE
pdm2_name=case

pdm2_aff1_name=elative
pdm2_aff1_orth=-ELAT
pdm2_aff2_name=nominative
pdm2_aff2_orth=NONE
pdm3_name=possessive
pdm3_aff1_name=1SG-POSS
pdm3_aff1_orth=-POSS1SG
pdm4_name=particle
pdm4_aff1_name=also
pdm4_aff1_orth=-kin
pdm5_name=pernum-agr
pdm5_aff1_name=1SG
pdm5_aff1_orth=-1SG
pdm5_aff2_name=3SG
pdm5_aff2_orth=-3SG
pdm5_aff3_name=indefinite
pdm5_aff3_orth=-INDEF
pdm6_name=tense
pdm6_aff1_name=past
pdm6_aff1_orth=-PAST
pdm6_aff2_name=nonpast
pdm6_aff2_orth=NONE
pdm7_name=mood
pdm7_aff1_name=conditional
pdm7_aff1_orth=-COND
pdm8_name=passive
pdm8_orth=-PASS

section=test-sentences
sentence1=opiskelija pidA-3SG omena-ELAT
sentence2=kAvele-1SG

A.4 Uzbek

version=2

section=language
language=Uzbek

section=word-order

word-order=sov
has-dets=no

section=sentential-negation

section=coordination

section=matrix-yes-no

section=inflection
morph1_type=pdm1
morph1_aff=suffix
morph1_opt=yes
morph1_dtr1_type=noun
morph2_type=pdm2
morph2_aff=suffix
morph2_opt=yes
morph2_dtr1_type=morph1
morph3_type=pdm3
morph3_aff=suffix
morph3_opt=no
morph3_dtr1_type=morph2
morph4_type=pdm4
morph4_aff=suffix
morph4_opt=yes
morph4_dtr1_type=verb
morph5_type=pdm5
morph5_aff=suffix
morph5_opt=no
morph5_dtr1_type=morph4
morph6_type=pdm6
morph6_aff=suffix
morph6_opt=yes
morph6_dtr1_type=morph5
morph7_type=pdm7
morph7_aff=suffix
morph7_opt=no
morph7_dtr1_type=morph6
morph8_type=pdm8
morph8_aff=suffix
morph8_opt=yes

morph8_dtr1_type=morph7

section=infl-paradigms
pdm1_name=plural
pdm1_orth=-PL
pdm2_name=possessive
pdm2_aff1_name=1SGPOS
pdm2_aff1_orth=-1SGPOS
pdm3_name=case
pdm3_aff1_name=accusative
pdm3_aff1_orth=-ACC
pdm3_aff2_name=nominative
pdm3_aff2_orth=NONE
pdm4_name=negation
pdm4_orth=-NEG
pdm5_name=TAM1
pdm5_aff1_name=nonpast
pdm5_aff1_orth=-NONPST
pdm5_aff2_name=past-perfective
pdm5_aff2_orth=-PSTPRF
pdm6_name=TAM2
pdm6_aff1_name=nonhabitual
pdm6_aff1_orth=-NONHAB
pdm7_name=subject-agr
pdm7_aff1_name=1SG
pdm7_aff1_orth=-1SG
pdm7_aff2_name=3SG
pdm7_aff2_orth=-3SG
pdm8_name=question
pdm8_orth=-QUES

section=basic-lexicon
noun1=oquwci
noun1_pred=_student_n_rel
noun1_det=imp
noun2=olma
noun2_pred=_apple_n_rel
noun2_det=imp
iverb=kel
iverb-pred=_come_v_rel
iverb-subj=np

tverb=ye
tverb-pred=_eat_v_rel
tverb-subj=np
tverb-obj=np

section=test-sentences
sentence1=kel-PSTPRF-1SG
sentence2=oquwci olma-ACC ye-NONPST-3SG

Appendix B

TEST SUITES

B.1 Zulu

Judgment: ungrammatical
NC-ntwana SC-TENSE-dlal-FV
“the child play”

Judgment: grammatical
C1-ntwana SC1-FUT-dlal-a
“The child will play”

Judgment: grammatical
C1-ntwana SC1-PRES-dlal-a
“The child plays”

Judgment: grammatical
C1-ntwana SC1-dlal-a
“The child plays”

Judgment: ungrammatical
ntwana SC1-PRES-dlal-a
“The child plays”

Judgment: ungrammatical
ntwana SC1-dlal-a
“The child plays”

Judgment: ungrammatical
ntwana PRES-dlal-a
“The child plays”

Judgment: ungrammatical
ntwana dlal-a
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-a
“The child plays”

Judgment: ungrammatical
C1-ntwana PRES-dlal-a
“The child plays”

Judgment: ungrammatical
ntwana-C1 SC1-PRES-dlal-a
“The child plays”

Judgment: ungrammatical
C1-ntwana PRES-SC1-dlal-a
“The child plays”

Judgment: ungrammatical
C1-ntwana PRES-a-SC1-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana PRES-SC1-a-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana a-PRES-SC1-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana a-SC1-PRES-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana SC1-a-PRES-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana SC1-PRES-a-dlal
“The child plays”

Judgment: ungrammatical

C1-ntwana a-SC1-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana SC1-a-dlal
“The child plays”

Judgment: ungrammatical
C1-ntwana SC1-dlal-PRES-a
“The child plays”

Judgment: ungrammatical
C1-ntwana PRES-dlal-SC1-a
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-PRES-SC1-a
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-a-PRES-SC1
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-PRES-a-SC1
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-a-SC1-PRES
“The child plays”

Judgment: ungrammatical
C1-ntwana dlal-SC1-a-PRES
“The child plays”

Judgment: ungrammatical
SC1-ntwana SC1-PRES-dlal-a
“The child plays”

Judgment: ungrammatical
PRES-ntwana SC1-PRES-dlal-a

“The child plays”

Judgment: ungrammatical
C1-ntwana SC1-FUT-OC-bon-a NC-yoka
“The child will see the snake”

Judgment: grammatical
C1-ntwana SC1-FUT-bon-a C9-yoka
“The child will see the snake”

Judgment: grammatical
C1-ntwana SC1-FUT-OC9-bon-a C9-yoka
“The child will see the snake”

Judgment: grammatical
C1-ntwana SC1-FUT-OC9-bon-a
“The child will see it”

Judgment: grammatical
SC1-FUT-OC9-bon-a C9-yoka
“He/she will see the snake”

Judgment: grammatical
SC1-FUT-OC9-bon-a
“He/she will see it”

Judgment: ungrammatical
C1-ntwana OC9-SC1-FUT-bon-a C9-yoka
“The child will see the snake”

Judgment: ungrammatical
C1-ntwana SC1-OC9-FUT-bon-a C9-yoka
“The child will see the snake”

Judgment: ungrammatical
C1-ntwana SC1-FUT-bon-OC9-a C9-yoka
“The child will see the snake”

Judgment: ungrammatical
C1-ntwana SC1-FUT-bon-a-OC9 C9-yoka
“The child will see the snake”

Judgment: ungrammatical
C1-ntwana SC1-FUT-OC9-bon-a OC9-yoka
“The child will see the snake”

Judgment: ungrammatical
C1-ntwana SC1-FUT-C9-bon-a C9-yoka
“The child will see the snake”

Judgment: grammatical
C1-ntwana NEG-SC1-PRES-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
NEG-C1-ntwana SC1-PRES-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
NEG-ntwana SC1-PRES-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
C1-ntwana-NEG SC1-PRES-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
C1-ntwana SC1-NEG-PRES-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
C1-ntwana SC1-PRES-NEG-dlal-e
“The child doesn’t play”

Judgment: ungrammatical
C1-ntwana SC1-PRES-dlal-NEG-e
“The child doesn’t play”

Judgment: ungrammatical
C1-ntwana SC1-PRES-dlal-e-NEG
“The child doesn’t play”

B.2 Slave

Judgment: grammatical
h-d-shin
1SG-d-shin
I sing.

Judgment: ungrammatical
d-shin-h
d-sing-1SG
I sing.

Judgment: grammatical
d-shin
d-sing
S/he sings.

Judgment: grammatical
ke-d-shin
3Pl.Human-d-sing
They sing.

Judgment: ungrammatical
d-shin-ke
d-sing-3PL.Human
They sing.

Judgment: grammatical
de-h-d-shin
INCEP-1SG-d-sing
I start to sing.

Judgment: ungrammatical
h-d-shin-de
1SG-d-sing-INCEP
I start to sing.

Judgment: ungrammatical
h-de-d-shin
1SG-INCEP-d-sing
I start to sing.

Judgment: grammatical
y-n-h-d-shin
yCONJ-PFV-1SG-d-sing
I sang.

Judgment: ungrammatical
n-y-h-d-shin
PFV-yCONJ-1SG-d-sing
I sang

Judgment: ungrammatical
y-h-n-d-shin
yCONJ-1SG-PVF-d-sing
I sang.

Judgment: ungrammatical
h-y-n-d-shin
1SG-yCONJ-PFV-d-sing
I sang.

Judgment: ungrammatical
h-y-d-shin-n
1SG-yCONJ-d-sing-PFV
I sang.

Judgment: ungrammatical
h-n-d-shin-y
1SG-PFV-d-sing-yCONJ
I sang.

Judgment: ungrammatical
h-d-shin-n-y
1SG-d-sing-PFV-yCONJ
I sang.

Judgment: ungrammatical
h-d-shin-y-n
1SG-d-sing-yCONJ-PFV
I sang.

Judgment: grammatical
y-n-d-shin
yCONJ-PFV-d-sing
S/He sang.

Judgment: grammatical
ke-de-y-n-d-shin
3PL.Human-INCEP-yCONJ-PFV-d-sing
They started to sing.

Judgment: ungrammatical
de-ke-y-n-d-shin
INCEP.3PL.Human-yCONJ-PFV-d-sing
They started to sing.

Judgment: ungrammatical
de-y-ke-n-d-shin
INCEP-yCONJ-3PL.Human-PFV-d-sing
They started to sing.

Judgment: ungrammatical
de-y-n-ke-d-shin
INCEP-yCONJ-PFV-3PL.Human-d-sing
They started to sing.

Judgment: ungrammatical
ke-y-de-n-d-shin
3PL.Human-yCONJ-INCEP-PFV-d-sing
They started to sing.

Judgment: ungrammatical
ke-y-n-de-d-shin
3PL.Human-yCONJ-PFV-INCEP-d-sing
They started to sing.

Judgment: ungrammatical
y-n-ke-de-d-shin
yCONJ-PVF-3PL.Human-INCEP-d-sing
They started to sing

Judgment: ungrammatical

y-n-de-ke-d-shin
yCONJ-PFV-INCEP-3PL.Human-d-sing
They started to sing.

Judgment: grammatical
na-h-d-shin
HAB-1SG-d-sing
I sing customarily/habitually.

Judgment: ungrammatical
h-na-d-shin
1SG-HAB-d-sing
I sing habitually.

Judgment: ungrammatical
h-d-shin-na
1SG-d-sing-HAB
I sing habitually.

Judgment: grammatical
du-h-d-shin
NEG-1SG-d-sing
I don’t sing.

Judgment: ungrammatical
h-du-d-shin
1SG-NEG-d-sing
I don’t sing.

Judgment: ungrammatical
h-d-shin-du
1SG-d-sing-NEG
I don’t sing.

Judgment: grammatical
na-be-ne-w-n-h-tah
THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her.

Judgment: grammatical
na-ne-w-n-h-tah

THM1-THM8-wCONj-PFV-1sg-move.foot
I kicked him/her.

Judgment: ungrammatical
ne-na-w-n-h-tah
THM8-THM1-wCONJ-PVF-h-move.foot
I kicked him/her.

Judgment: ungrammatical
na-w-ne-n-h-tah
THM1-wCONJ-THM8-PFV-1SG-move.foot
I kicked him/her.

Judgment: ungrammatical
na-w-n-ne-h-tah
THM1-wCONJ-PFV-THM8-1SG-move.foot
I kicked him/her.

Judgment: ungrammatical
na-w-n-h-ne-tah
THM1-wCONJ-PFV-1SG-THM8-move.foot
I kicked him/her

Judgment: ungrammatical
na-w-n-h-tah-ne
THM1-wCONJ-PFV-1SG-THM8-move.foot
I kicked him/her.

Judgment: ungrammatical
be-na-ne-w-n-h-tah
3SG.Human-THM1-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her.

Judgment: ungrammatical
na-ne-be-w-n-h-tah
THM1-THM8-3SG.Human-wCONJ-PFV-1SG-move.foot
I kicked him/her.

Judgment: ungrammatical
na-ne-w-be-n-h-tah
THM1-THM8-wCONJ-3SG.Human-PFV-1SG-move.foot

I kicked him/her.

Judgment: ungrammatical
na-ne-w-n-be-h-tah
THM1-THM8-wCONJ-PFV-3SG.Human-1SG-move.foot
I kicked him/her.

Judgment: ungrammatical
na-ne-w-n-h-be-tah
THM1-THM8-wCONJ-PFV-1SG-3SG.Human-move.foot
I kicked him/her.

Judgment: ungrammatical
na-ne-w-n-h-tah-be
THM1-THM8-wCONJ-PFV-1SG-move.foot-3SG.Human
I kicked him/her.

Judgment: grammatical
na-ele-ke-ne-w-n-d-tah
THM1-RECP-3PL.Human-THM8-wCONJ-PFV-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
ele-na-ke-ne-w-n-d-tah
RECP-THM1-3PL.Human-THM8-wCONJ-PFV-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ele-ne-w-n-d-tah
THM1-3PL.Human-RECP-THM8-wCONJ-PFV-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ne-ele-w-n-d-tah
THM1-3PL.Human-THM8-RECP-wCONJ-PFV-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ne-w-ele-n-d-tah
THM1-3PL.Human-THM8-wCONJ-RECP-PFV-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ne-w-n-ele-d-tah
THM1-3PL.Human-THM8-wCONJ-PFV-RECP-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ne-w-n-d-ele-tah
THM1-3PL.Human-THM8-wCONJ-PFV-RECP-RECP-move.foot
They kicked each other.

Judgment: ungrammatical
na-ke-ne-w-n-d-tah-ele
THM1-3PL.Human-THM8-wCONJ-PFV-RECP-move.foot-RECP
They kicked each other.

Judgment: grammatical
na-leh-ele-ne-w-n-d-tah
THM1-DU-RECP-THM8-wCONJ-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-leh-ele-ke-ne-w-n-d-tah
THM1-DU-RECP-3PL.Human-THM8-wCONJ-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
leh-na-ele-ne-w-n-d-tah
DU-THM1-RECP-THM8-wCONJ-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-leh-ne-w-n-d-tah
THM1-RECP-DU-THM8-wCONJ-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-ne-leh-w-n-d-tah
THM1-RECP-THM8-DU-wCONJ-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-ne-w-leh-n-d-tah
THM1-RECP-THM8-wCONJ-DU-PFV-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-ne-w-n-leh-d-tah
THM1-RECP-THM8-wCONJ-PFV-DU-RECP-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-ne-w-n-d-leh-tah
THM1-RECP-THM8-wCONJ-PFV-RECP-DU-move.foot
They two kicked each other.

Judgment: ungrammatical
na-ele-ne-w-n-d-tah-leh
THM1-RECP-THM8-wCONJ-PFV-RECP-move.foot-DU
They two kicked each other.

Judgment: grammatical
na-dlo-be-ne-w-n-h-tah
THM1-laugh-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
dlo-na-be-ne-w-n-h-tah
laugh-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
na-be-dlo-ne-w-n-h-tah
THM1-3SG.Human-laugh-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
na-be-ne-dlo-w-n-h-tah
THM1-3SG.Human-THM8-laugh-wCONJ-PFV-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical

na-be-ne-w-dlo-n-h-tah
THM1-3SG.Human-THM8-wCONJ-laugh-PFV-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
na-be-ne-w-n-dlo-h-tah
THM1-3SG.Human-THM8-wCONJ-PFV-dlo-1SG-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
na-be-ne-w-n-h-dlo-tah
THM1-3SG.Human-THM8-wCONJ-PFV-1SG-dlo-move.foot
I kicked him/her while laughing.

Judgment: ungrammatical
na-be-ne-w-n-h-tah-dlo
THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot-laugh
I kicked him/her while laughing.

Judgment: grammatical
na-ya-be-ne-i-w-n-h-tah
THM1-DISTR-3SG.Human-THM8-SER-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
i-na-ya-be-ne-w-n-h-tah
SER-THM1-DISTR-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-i-ya-be-ne-w-n-h-tah
THM1-SER-DISTR-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-i-be-ne-w-n-h-tah
THM1-DISTR-SER-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-be-i-ne-w-n-h-tah

THM1-DISTR-3SG.Human-SER-THM8-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-be-ne-w-i-n-h-tah
THM1-DISTR-3SG.Human-THM8-wCONJ-SER-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-be-ne-w-n-i-h-tah
THM1-DISTR-3SG.Human-THM8-wCONJ-PFV-SER-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-be-ne-w-n-h-i-tah
THM1-DISTR-3SG.Human-THM8-wCONJ-PFV-1SG-SER-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-ya-be-ne-w-n-h-tah-i
THM1-DISTR-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot-SER
I kicked him/her repeatedly.

Judgment: ungrammatical
ya-na-be-ne-i-w-n-h-tah
DISTR-THM1-3SG.Human-THM8-SER-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ya-ne-i-w-n-h-tah
THM1-3SG.Human-DISTR-THM8-SER-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-ya-i-w-n-h-tah
THM1-3SG.Human-THM8-DISTR-SER-wCONJ-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-i-ya-w-n-h-tah
THM1-3SG.Human-THM8-SER-DISTR-wCONJ-PFV-1SG-move.foot

I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-i-w-ya-n-h-tah
THM1-3SG.Human-THM8-SER-wCONJ-DISTR-PFV-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-i-w-n-ya-h-tah
THM1-3SG.Human-THM8-SER-wCONJ-PFV-DISTR-1SG-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-i-w-n-h-ya-tah
THM1-3SG.Human-THM8-SER-wCONJ-PFV-1SG-DISTR-move.foot
I kicked him/her repeatedly.

Judgment: ungrammatical
na-be-ne-i-w-n-h-tah-ya
THM1-3SG.Human-THM8-SER-wCONJ-PFV-1SG-move.foot-DISTR
I kicked him/her repeatedly.

Judgment: grammatical
be-keh-na-be-ne-w-n-h-tah
3SG-into-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-n-h-tah
3SG-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-w-n-h-tah
into-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-be-na-be-ne-w-n-h-tah
into-3SG-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-be-ne-w-n-h-tah
into-THM1-3SG-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-be-w-n-h-tah
into-THM1-3SG.Human-THM8-3SG-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-w-be-n-h-tah
into-THM1-3SG.Human-THM8-wCONJ-3SG-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-w-n-be-h-tah
into-THM1-3SG.Human-THM8-wCONJ-PFV-3SG-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-w-n-h-be-tah
into-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-3SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
keh-na-be-ne-w-n-h-tah-be
into-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot-3SG
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-n-h-tah
3SG-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-keh-be-ne-w-n-h-tah
3SG-THM1-into-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-keh-ne-w-n-h-tah
3SG-THM1-3SG.Human-into-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-keh-w-n-h-tah
3SG-THM1-3SG.Human-THM8-into-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-keh-n-h-tah
3SG-THM1-3SG.Human-THM8-wCONJ-into-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-n-keh-h-tah
3SG-THM1-3SG.Human-THM8-wCONJ-PFV-into-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-n-h-keh-tah
3SG-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-into-move.foot
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
be-na-be-ne-w-n-h-tah-keh
3SG-THM1-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot-into
I kick him/her into it (e.g. hole).

Judgment: ungrammatical
na-be-keh-be-ne-w-n-h-tah
THM1-3SG-into-3SG.Human-THM8-wCONJ-PFV-1SG-move.foot
I kick him/her into it (e.g. hole).

Judgment: grammatical
du-be-keh-na-ya-dlo-leh-ele-ne-i-w-n-id-d-tah
NEG-3SG-into-THM-DISTR-laugh-DU-RECP-THM-SER-wCONJ-PFV-1DU-RECP-move.foot
We two didn’t kick each other into it (e.g. a hole) repeatedly while laughing.

B.3 Finnish

Judgment: grammatical
opiskelija pidA-3SG omena-ELAT
student(NOM) like-3SG apple-ELAT
“The student likes the apple”

Judgment: ungrammatical
opiskelija 3SG-pidA ELAT-omena
student(NOM) like-3SG apple-ELAT
“The student likes the apple”

Judgment: ungrammatical
opiskelija pidA-ELAT omena-3SG
student(NOM) like-3SG apple-ELAT
“The student likes the apple”

Judgment: grammatical
opiskelija pidA-3SG omena-PL-ELAT
student(NOM) like-3SG apple-PL-ELAT
“The student likes the apples”

Judgment: ungrammatical
opiskelija pidA-3SG omena-ELAT-PL
student(NOM) like-3SG apple-ELAT-PL
“The student likes the apples”

Judgment: ungrammatical
opiskelija pidA-3SG PL-omena-ELAT
student(NOM) like-3SG PL-apple-ELAT
“The student likes the apples”

Judgment: ungrammatical
opiskelija pidA-3SG-PL omena-ELAT
student(NOM) like-3SG-PL apple-ELAT
“The student likes the apples”

Judgment: ungrammatical
opiskelija pidA-PL-3SG omena-ELAT
student(NOM) like-PL-3SG apple-ELAT
“The student likes the apples”

Judgment: grammatical
opiskelija pidA-3SG omena-PL-ELAT-POSS1SG
student(NOM) like-3SG apple-Pl-ELAT-POSS1SG
“The student likes my apples”

Judgment: ungrammatical
opiskelija pidA-3SG omena-PL-POSS1SG-ELAT
student(NOM) like-3SG apple-Pl-POSS1SG-ELAT
“The student likes my apples”

Judgment: ungrammatical
opiskelija pidA-3SG omena-POSS1SG-PL-ELAT
student(NOM) like-3SG apple-POSS1SG-PL-ELAT
“The student likes my apples”

Judgment: grammatical
opiskelija pidA-3SG omena-ELAT-POSS1SG
student(NOM) like-3SG apple-ELAT-POSS1SG
“The student likes my apple”

Judgment: ungrammatical
opiskelija pidA-3SG-POSS1SG omena-ELAT
student(NOM) like-3SG-POSS1SG apple-ELAT
“The student likes my apple”

Judgment: ungrammatical
opiskelija pidA-POSS1SG-3SG omena-ELAT
student(NOM) like-POSS1SG-3SG apple-ELAT
“The student likes my apple”

Judgment: grammatical
opiskelija pidA-3SG omena-ELAT-POSS1SG
student(NOM) like-3SG apple-ELAT-POSS1SG
“The student likes my apple”

Judgment: grammatical
opiskelija pidA-3SG omena-PL-ELAT-POSS1SG-kin
student like-3SG apple-PL-ELAT-POSS1SG-also
“The student likes my apples also (in addition to liking other things).”

Judgment: grammatical
opiskelija kAvele-3SG
student(NOM) walk-3SG
“The student walks”

Judgment: grammatical
kAvele-1SG
walk-1SG
“I walk”

Judgment: grammatical
kAvele-PAST-1SG
walk-PAST-1SG
“I walked”

Judgment: ungrammatical
kAvele-1SG-PAST
walk-1SG-PAST
“I walked”

Judgment: ungrammatical
kAvele-PAST
walk-PAST
“I walked”

Judgment: grammatical
kAvele-COND-1SG
walk-COND-1SG
“I would walk”

Judgment: ungrammatical
kAvele-1SG-COND
walk-1SG-COND
“I would walk”

Judgment: ungrammatical
kAvele-COND
walk-COND
“I walked”

Judgment: ungrammatical

kAvele-COND-PAST-1SG
walk-COND-PAST-1SG
“I would have walked”

Judgment: ungrammatical
kAvele-PAST-COND-1SG
walk-PAST-COND-1SG
“I would have walked”

Judgment: grammatical
kAvele-PASS-INDEF
walk-PASS-INDEF
“One walks”

Judgment: grammatical
kAvele-PASS-PAST-INDEF
walk-PASS-PAST-INDEF
“One walked”

Judgment: grammatical
kavele-PASS-COND-INDEF
walk-PASS-COND-INDEF
“One would walk”

Judgment: grammatical
kavele-PASS-COND-INDEF-kin
walk-PASS-COND-INDEF-also
“One would also walk”

Judgment: ungrammatical
kavele-COND-PASS-INDEF-kin
walk-COND-PASS-INDEF-also
“One would also walk”

Judgment: ungrammatical
kavele-COND-INDEF-PASS-kin
walk-COND-INDEF-PASS-also
“One would also walk”

Judgment: ungrammatical
kavele-COND-INDEF-kin-PASS

walk-COND-INDEF-also-PASS
“One would also walk”

Judgment: ungrammatical
kavele-PASS-COND-kin-INDEF
walk-PASS-COND-also-INDEF
“One would also walk”

Judgment: ungrammatical
kavele-PASS-kin-COND-INDEF
walk-PASS-also-COND-INDEF
“One would also walk”

Judgment: ungrammatical
kavele-also-PASS-COND-INDEF
walk-also-PASS-COND-INDEF
“One would also walk”

B.4 Uzbek

Judgment: grammatical
kel-PSTPRF-1SG
come-PST.PRF-1SG
”I came, I have come”

Judgment: ungrammatical
1SG-kel-PSTPRF
1SG-come-PST.PRF
”I came, I have come”

Judgment: ungrammatical
kel-1SG-PSTPRF
come-1SG-PST.PRF
”I came, I have come”

Judgment: ungrammatical
PSTPRF-kel-1SG
PSTPRF-come-1SG
”I came, I have come”

Judgment: ungrammatical
kel-1SG
come-1SG
”I came, I have come”

Judgment: grammatical
oquwci kel-PSTPRF-3SG
student come-PST.PRF-3SG
”The student came”

Judgment: ungrammatical
oquwci-3SG kel-PSTPRF
student come-PST.PRF
”The student came”

Judgment: ungrammatical
oquwci-PSTPRF kel-3SG
student-PST.PRF come-3SG
”The student came”

Judgment: grammatical
kel-NEG-PSTPRF-1SG
come-NEG-PST.PRF-1SG
”I didn’t come”

Judgment: ungrammatical
NEG-kel-PSTPRF-1SG
NEG-come-PST.PRF-1SG
”I didn’t come”

Judgment: ungrammatical
kel-PSTPRF-NEG-1SG
come-PST.PRF-NEG-1SG
”I didn’t come”

Judgment: ungrammatical
kel-PSTPRF-1SG-NEG
come-PST.PRF-1SG-NEG
”I didn’t come”

Judgment: grammatical

kel-NEG-PSTPRF-1SG-QUES
come-NEG-PST.PRF-1SG-QUES
”Didn’t I come?”

Judgment: ungrammatical
QUES-kel-NEG-PSTPRF-1SG
QUES-come-NEG-PST.PRF-1SG
”Didn’t I come?”

Judgment: ungrammatical
kel-QUES-NEG-PSTPRF-1SG
come-QUES-NEG-PST.PRF-1SG
”Didn’t I come?”

Judgment: ungrammatical
kel-NEG-QUES-PSTPRF-1SG
come-NEG-QUES-PST.PRF-1SG
”Didn’t I come?”

Judgment: ungrammatical
kel-NEG-PSTPRF-QUES-1SG
come-NEG-PST.PRF-QUES-1SG
”Didn’t I come?”

Judgment: grammatical
oquwci kel-NEG-PSTPRF-3SG
student come-NEG-PST.PRF-3SG
”The student didn’t come”

Judgment: ungrammatical
oquwci-NEG kel-PSTPRF-3SG
student come-PST.PRF-3SG
”The student didn’t come”

Judgment: grammatical
oquwci kel-PSTPRF-3SG-QUES
student come-PST.PRF-3SG-QUES
”Did the student come?”

Judgment: ungrammatical
oquwci-QUES kel-PSTPRF-3SG

student come-PST.PRF-3SG
”Did the student come?”

Judgment: grammatical
oquwci olma-ACC ye-NONPST-3SG
student apple-ACC eat-NONPST-3SG
”The student eats/will eat the apple”

Judgment: ungrammatical
oquwci ACC-olma ye-NONPST-3SG
student ACC-apple eat-NONPST-3SG
”The student eats/will eat the apple”

Judgment: ungrammatical
oquwci olma ye-NONPST-3SG-ACC
student apple eat-NONPST-3SG-ACC
”The student eats/will eat the apple”

Judgment: ungrammatical
oquwci olma ye-NONPST-ACC-3SG
student apple eat-NONPST-ACC-3SG
”The student eats/will eat the apple”

Judgment: ungrammatical
oquwci olma ye-ACC-NONPST-3SG
student apple eat-ACC-NONPST-3SG
”The student eats/will eat the apple”

Judgment: ungrammatical
oquwci olma ACC-ye-NONPST-3SG
student apple ACC-eat-NONPST-3SG
”The student eats/will eat the apple”

Judgment: grammatical
oquwci olma-PL-ACC ye-NONPST-3SG
student apple-PL-ACC eat-NONPST-3SG
”The student eats/will eat the apples”

Judgment: grammatical
oquwci-PL olma-ACC ye-NONPST-3SG
student-PL apple-ACC eat-NONPST-3SG

”The students eat/will eat the apple”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-NONPST-3SG-PL
student apple-PL-ACC eat-NONPST-3SG-PL
”The student eats/will eat the apples”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-NONPST-PL-3SG
student apple-PL-ACC eat-NONPST-PL-3SG
”The student eats/will eat the apples”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-PL-NONPST-3SG
student apple-PL-ACC eat-PL-NONPST-3SG
”The student eats/will eat the apples”

Judgment: ungrammatical
oquwci olma-PL-ACC PL-ye-NONPST-3SG
student apple-PL-ACC PL-eat-NONPST-3SG
”The student eats/will eat the apples”

Judgment: grammatical
oquwci olma-PL-1SGPOS-ACC ye-NONPST-3SG
student apple-PL-1SG.POS-ACC eat-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci 1SGPOS-olma-PL-ACC ye-NONPST-3SG
student 1SG.POS-apple-PL-ACC eat-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-1SGPOS-PL-ACC ye-NONPST-3SG
student apple-1SG.POS-PL-ACC eat-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-PL-ACC-1SGPOS ye-NONPST-3SG
student apple-PL-ACC-1SG.POS eat-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-PL-ACC 1SGPOS-ye-NONPST-3SG
student apple-PL-ACC 1SG.POS-eat-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-1SGPOS-NONPST-3SG
student apple-PL-ACC eat-1SG.POS-NONPST-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-NONPST-1SGPOS-3SG
student apple-PL-ACC eat-NONPST-1SG.POS-3SG
”The student eats/will eat my apples”

Judgment: ungrammatical
oquwci olma-PL-ACC ye-NONPST-3SG-1SGPOS
student apple-PL-ACC eat-NONPST-3SG-1SGPOS
”The student eats/will eat my apples”

Judgment: grammatical
oquwci olma-ACC ye-NONPST-NONHAB-3SG
student apple-ACC eat-NONPST-NONHAB-3SG
”The student is eating the apple”

Judgment: ungrammatical
oquwci olma-ACC NONHAB-ye-NONPST-3SG
student apple-ACC NONHAB-eat-NONPST-3SG
”The student is eating the apple”

Judgment: ungrammatical
oquwci olma-ACC ye-NONPST-3SG-NONHAB
student apple-ACC eat-NONPST-3SG-NONHAB
”The student is eating the apple”

Judgment: grammatical
oquwci olma-PL-1SGPOS-ACC ye-NEG-NONPST-NONHAB-3SG-QUES
student apple-PL-1SG.POS-ACC eat-NEG-NONPST-NONHAB-3SG-QUES
”Isn’t the student eating the apple?”

Appendix C

SAMPLE RULE HIERARCHY

This is rule hierarchy for the section of the Zulu grammar relevant to the lexical rule types. Any parts of the grammar not created or changed by the morphotactic system are omitted. Also please note that this is the hierarchy from the grammar generated by the sample choices file, which was used to model various morphotactic phenomena for testing purposes. As such, it was not necessary to include the full paradigm for every morpheme slot. A complete grammar for Zulu would have a much more extensive hierarchy.