Automated Grammar Engineering for Verbal Morphology

David Wax

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

Master of Science

University of Washington

2014

Reading Committee:

Emily M. Bender, Chair

Fei Xia

Program Authorized to Offer Degree:
Department of Linguistics

University of Washington

Abstract

Automated Grammar Engineering
for Verbal Morphology

David Wax

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

This study examines the cross-linguistic potential for the automatic analysis of verbal morphology and creation of implemented formal grammars using the Grammar Matrix customization system and the rich linguistic resource of interlinear glossed text (IGT). Each grammar is automatically customized from the LinGO Grammar Matrix core grammar, with a focus on the morphotactics, morphosyntax, and morphosemantics of verbal forms for the target languages. Each step of the process is evaluated for two low-resource languages, Turkish and Tagalog, with data from the Online Database of Interlinear Glossed Text, and over a specially curated set of IGT of the French verb *faire*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0644097. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Chapter 1

INTRODUCTION

A machine-interpretable grammar is a useful tool for studying languages as well as testing linguistic hypotheses (Bender, 2008). The difficulty comes from the amount of time and knowledge that goes into making a grammar which can be used both by humans and computers. A human engineered grammar can capture complex analyses and phenomena, but building grammars by hand is a very time consuming process, and there are too many languages in the world for each to be feasibly dealt with individually.

This work is part of a larger initiative to to help expedite the process of building starter Head-driven Phrase Structure Grammars (HPSG) (Pollard & Sag, 1994) for use with computer parsers and generators. This is accomplished by using data mined from linguistically rich resources, specifically from the Online Database of Interlinear Text (Lewis, 2006) from the RiPLes[1] project. By using these rich sources of information, even smaller languages which may not have enough resources for most machine learning techniques can be processed to create working grammars. The deep linguistic information contained in the Grammar Matrix core grammar contributes the framework needed for making working grammars as well as providing the general analysis which allows this process to work for any language with even a small amount of IGT available. By utilizing the Grammar Matrix questionnaire (Bender et al., 2002, 2010a) in conjunction with these linguistic information sources, it is possible to automate the creation of working HPSG grammars for parsing and generation by programs such as the Linguistic Knowledge Builder (LKB) (Copestake, 2002).

This study explores the amount of data required to build such HPSG grammars and

[1] `http://faculty.washington.edu/fxia/riples/`

measures the effectiveness depending on the amounts of data available, while focusing on ways to work with low resource and endangered languages. It specifically looks at constructing the rules of verbal morphology from examples of interlinear text by collecting the data needed by the Grammar Matrix questionnaire morphology form to make a compilable starter HPSG grammar. The process uses resources for working with English as well as the consistent structure of IGT to determine and interpret the information contained throughout in the rest of each example.

Two different sources of data were used in this study to test an idealistic potential as well as the current working potential of this process. The first is a complete paradigm for a single phonetically regularized French verb where each gloss corresponds to one language morpheme, and the second using the data drawn from ODIN, which is more naturally occurring and noisy data.

In the case of the French data, having a complete and known set of verb forms which followed very strict glossing rules allowed for clear control on the amount of data provided to the system as well as a large amount of data for evaluation. By iteratively reducing the size of the training data set, the potential of the system to work with cases of data sparsity could be examined.

The ODIN data is used to evaluate the system's potential to utilize real world data and its ability to work with languages which do not have an abundance of linguistic resources. In addition, these tests provide a better understanding of some of the difficulties of automated extraction and the system's ability to deal with noisy data.

This thesis first discusses the background of this work in Chapter 2 by talking about the projects which contribute to the system, while also looking at some similar projects and how they vary from this study. Next it provides information about the steps necessary for getting from a database of IGT to a compilable grammar in Chapter 3. Chapter 4 discusses the parameters used during these steps for modifying the processes, as well as providing a closer look at each data set, the construction of the gold standard they are evaluated against, and the parameters used and tested when working with each data set. After that Chapter

5 looks deeper into the inner working of how the data is stored and the classes used during the process, as well as further details concerning how this data interfaces with external programs. The results are reported in Chapter 6, examining the experiments used for both data sets and the benefits of certain techniques usable by the system. Finally Chapter 7 considers the implications of this work such as the extraction of information from IGT for learning about noun forms or sentence structure, and other potential areas to improve.

Chapter 2
BACKGROUND

This section outlines the different projects used by this study and presents details for the resources used and generated by the system. It starts by looking at the core components, followed by the auxiliary programs. Finally, an overview is given of past projects with similar goals and the similarities and differences of those projects.

2.1 LinGO Grammar Matrix

The LinGO Grammar Matrix (Bender et al., 2002, 2010a) is used in the development of HPSG grammars as a part of DELPH-IN (Deep Linguistic Processing with HPSG Initiative) [http://www.delph-in.net]. The Grammar Matrix was first developed from the English Resource Grammar (Flickinger, 2000) and the Jacy grammar of Japanese (Siegel & Bender, 2002), by extracting the crosslinguistic types and constraints. It provides a core grammar to be used as a starting point for further HPSG grammar development. Further language specific information is created around the core grammar, allowing for an agile creation of language specific grammars. The Grammar Matrix has been used since 2004 in a course on multilingual grammar engineering, which has led the development of many grammar fragments based on written grammars and references (Bender, 2007). Later libraries were created to allow for the common but non-universal language properties to be easily accessed by grammar developers. These libraries became the basis of the customization system[1] (Bender & Flickinger, 2005; Drellishak, 2009; Bender et al., 2010b), which provides a set of linguistically motivated analyses which can be applied to many different languages. Users answer questions in regards to the language, then the Grammar Matrix customization sys-

[1] Accesible at `www.delph-in.net/matrix/customize`

tem uses choices made by the user and the customization libraries to build a compilable starter grammar fragment. These grammars provide a great starting point for further hand development and hypothesis testing.

2.1.1 Verb Types and Position Classes

Verb types and position classes make up the backbone of the verbal morphology of the LinGO Grammar Matrix customization system. This section covers these two terms from a high level and gives some examples of how they are used. Further detail on how they are represented in the LinGO Grammar Matrix is provided in Subsection 2.1.2. Verb types and position classes have a lot of flexibility which is not fully utilized by this study which instead has a focus on just general cases which should be true for most languages.

Example (1) shows the definition of a verb type. Verb types are used to encapsulate sets of verbal roots and the grammatical qualities contained by the verb roots. Each verb type is defined by its transitivity and the syntactic and semantic features of the root of the verb. While the rules of inflection are not defined in the verb type, the verb type is used to group verbs by the way they inflect. The rules concerning which verbs can take which affixes is dictated by the inputs of the position class. Inputs are explained later in this subsection.

(1) Verb Type: A verb type consists a signature and a set of instances. The signature is comprised of a set of features and one of the values *transitive* or *intransitive* for the valence. Each instance consists of a verb root spelling and predicate value.

In the following set of examples (2-6) there are two possible verb types would be one for the regular verbs and one for the irregular past verbs. The regular verb type, say *verb1*, would be used for verbs (2), (3), (4) would be defined as an intransitive verb and have no features defined. Even though there is a tense defined on the verbs from (2) and (3) and agreement on the verb from (4), the meaning is related to the suffix not the verb root itself. Features related to affixes will be used with the position classes and explained later in this subsection. The second verb type, *verb2*, for irregular past verbs from (5) and (6) would

have a transitive valance value. This verbs also have past tense as part of the verb root, so the verb type which describes them will also have a tense feature defined.

(2) *walk-ed*
walk-PAST

(3) *sleep-ed*
sleep-PAST

(4) *dance-s*
dance-3.sg

(5) *sat*
sit.PAST

(6) *forgot*
forget.PAST

In general, in the Grammar Matrix (and in HPSG) verb types can be used to capture properties beyond inflectional classes (e.g. valence frames) and can also be used to capture cross-cutting generalizations, where verb types inherit from other (partially specified) types. In this work, however, verb types will only be used to capture (a) inflectional classes and (b) sets of verbs that have morphosyntactic features associated directly with the root.

Affixes are described in position classes, and position classes have different defining qualities than verb types. The definition of a position class is shown in Example (7). Each position class represents a set of prefixes or suffixes but never both. Each position class also has a list of *inputs*; these inputs represent which verb types or other position classes an affix may attach to. The affixes seen in verbs (2), (3), and (4) can all be represented by the same position class, *verb-pc1*. Each affix is from those verbs is a suffix and attaches to the same verb type, *verb1*. Unlike verb types there is no restriction to only being limited to a unique set of features. Each instance of an affix defined by a position class will contain the spelling paired with the features it represents. In cases where there is a null morpheme, the spelling is left as an empty string but still be paired with specific features.

(7) Position Class: A position class consists of three parts, a type of affix, a list of inputs, and lexical rule types. The type of affix is either *prefix* or *suffix*. The list of inputs consists of a set of verb types and/or other position classes. The lexical rule types defines a set of features connected affix spellings. The affix spelling can be left as an empty string.

Position classes also may contain any number of inputs, and inputs are not limited to verb types, but can also be other position classes. Consider the verb shown in (8); *verb-pc1* already defines a suffix *ed* with the meaning *PAST*. A second position, *verb-pc2*, is needed to define the prefix *un*. *Verb-pc2* would be defined as a suffix, and have an spelling/feature pair with *un/NEG*. It's input is not a verb type, but another position class, specifically *verb-pc1*. Inputs establish the order in which affixes attach; inputs do not dictate which morpheme they attach to. So even though the prefix *un* attaches to the verb root, because it attaches after the suffix, the position class with the suffix is the prefixes input.

(8) *un-walk-ed*
NEG-walk-PAST

This section covered the concept of verb types and position classes from a very high level. The next section provides more information to the specifics of how these two terms are concretely defined, and why they are important to this study.

2.1.2 Choices File

The LinGO Grammar Matrix customization system stores the answers to the questionnaire needed to build a grammar in a plain text file referred to as a *choices file*. The choices file is an important part of the MOM system, because it is the target output format for every test. There are two sections which are of primary interest: Lexicon and Morphology.

Figure 2.1.2 shows an example of the lexicon section in a choices file. Verb1 shows an example of a verb type. Features and valency are defined for each verb type. For example verb1 has two features specified, number singular and first person. The minimum amount

```
section=lexicon
   verb1_valence=trans
      verb1_feat1_name=number
      verb1_feat1_value=singular
      verb1_feat1_head=verb
      verb1_feat2_name=person
      verb1_feat2_value=1st
      verb1_feat2_head=verb
      verb1_stem1_orth=ayoko
      verb1_stem1_pred=_dislike_v_rel
   verb2_valence=trans
      verb2_stem1_orth=bumili
      verb2_stem1_pred=_bought_v_rel
      verb2_stem2_orth=lutu
      verb2_stem2_pred=_cook_v_rel
```

Figure 2.1: Lexicon Section from an Example Choices File

required to define a verb is providing an orthography and predicate value and a verb type with a transitive or intransitive valence. Multiple verb orthography and predicate pairs can be provided for a single verb type, which all have the same features present, as seen in verb2 from Figure 2.1.2.

The morphology section is more complicated. A position class is created for each affix slot and is defined as either a suffix or a prefix. A position class basically represents a slot where one of a set of affixes can be attached. The order of position classes is defined by the inputs of each position class. Specifically, these inputs define which uninflected verb instances and verbs inflected by another position class the affix may attach to. For example verb-pc1 from Figure 2.1.2 is a prefix which attaches to any verbs in the verb2 class from the lexicon section. A position class can also take other position classes as seen in verb-pc2, or even take multiple verb types and other position classes.

Position classes are further divided into lexical rule types. The lexical rule types provide

```
section=morphology
   verb-pc1_order=prefix
   verb-pc1_inputs=verb2
      verb-pc1_lrt1_feat1_name=asp
      verb-pc1_lrt1_feat1_value=asp_T
      verb-pc1_lrt1_feat1_head=verb
      verb-pc1_lrt1_lri1_inflecting=yes
      verb-pc1_lrt1_lri1_orth=lu
      verb-pc2_lrt2_lri1_inflecting=verb
      verb-pc2_lrt2_lri1_orth=pan
   verb-pc2_order=suffix
   verb-pc2_inputs=verb_pc1
      verb-pc2_lrt1_feat1_name=case
      verb-pc2_lrt1_feat1_value=acc
      verb-pc2_lrt1_feat1_head=obj
      verb-pc2_lrt1_lri1_inflecting=yes
      verb-pc2_lrt1_lri1_orth=in
```

Figure 2.2: Morphology Section from an Example Choices File

the features that will be added by a lexical rule instance. Lexical rule types which occur in the same position class don't have to have the same features. Finally, lexical rule instances provide the orthography for the specific affix; although phonologically null morphemes are allowed, which are lexical rule instances without orthographies.

The choices files is format can easily be modified using the Grammar Matrix questionnaire. A grammar engineer can improve the grammar by modifying the generated Type Definition Language (tdl) files created by the Grammar Matrix customization system for further grammar development.

The resulting definitions for the verb-pc1 position class, its lexical rule types, and its lexical rule instances from Figure 2.1.2 in tdl would look like:

(9)
```
verb-pc1-lex-rule-super := add-only-no-ccont-rule & infl-lex-rule &
  [ DTR verb2-verb-lex ].
```

```
verb-pc1_lrt1-lex-rule := verb-pc1-lex-rule-super &
  [ SYNSEM.LOCAL.CAT.HEAD.ASP ASP_T ].

verb-pc1_lrt2-lex-rule := verb-pc1-lex-rule-super.

verb-pc1_lrt1-prefix :=
%prefix (* lu)
verb-pc1_lrt1-lex-rule.

verb-pc1_lrt2-prefix :=
%prefix (* pan)
verb-pc1_lrt2-lex-rule.
```

The choices file is a concise manner of defining the qualities of a grammar, the interface to the Grammar Matrix customization system, and the output of the MOM system described in this thesis. By automatically creating choices files from IGT, this system aims to speed up the grammar engineering process.

2.2 IGT

Interlinear glossed text (IGT) is a format for presenting linguistic data, commonly seen in linguistic papers around the world. The basic structure is three lines of text such as:

(10) *Lu-lutu-in* *ng* *lalaki* *ang* *adobo.*
Asp-cook-Acc CS man ANG adobo
'The man will cook the adobo.' (Rackowski & Richards, 2005) [tgl]

The top line is a line from the language being described, referred to in this paper as the language line. In this case, the language line is a simple sentence in Tagalog. The language line is typically morpheme segmented. The middle line provides a gloss for the language line, providing a gloss for each segmented morpheme. The final line is a translation of the original language line, in our example the translation is into English.

Even a single example of IGT from a linguistic paper can provide a wealth of linguistic information, especially when that IGT follows a clear standard such as the most widely

used Leipzig Glossing Rules (Bickel et al., 2008). IGT presents information with ease which makes it a very powerful and commonly used means of providing information to a reader.

IGT plays a critical role in the MOM system, because it is the format for all the training data. This study examines how well computers can take advantage of the richness of information found in IGT.

2.3 ODIN

The Online Database of Interlinear Text (ODIN) is a database of IGT extracted from linguistic papers available on the web (Lewis, 2006). The ODIN project is a part of the greater effort of the GOLD Community of Practice (Farrar & Lewis, 2007) designed to promote best practice standards for linguistic descriptions. The goal of ODIN is to provide a developed infrastructure for the searching of linguistic information. ODIN allows the data to be queried by the name of the language or the ISO code and has examples of IGT for hundreds of languages and thousands of instances.

Xia and Lewis (2007) used this data to automatically determine the predominant word order of sentences, and they tested the process on seven different languages. Because there are many tools available for processing and understanding English text, the English line can be a rich source of linguistic information. This information from the English lines was projected to the translation lines through the gloss line for providing a more accurate alignment than just trying to align the translation line with the language line directly. Their work has demonstrated that accurate information can be extracted from limited linguistic data. Due to the sparsity of recourses for many languages, leveraging data from all three lines of IGT is necessary for such a process.

IGT from ODIN in the input to the system for this study, therefore ODIN plays an important part as a source of linguistics information during this study and for future work.

2.4 GIZA++

GIZA was created in as a part of the statistical machine translation toolkit, Egypt[2]. GIZA++ (Och & Ney, 2003) is an extension of the original GIZA program, both being designed for use to train statistical translation models from bilingual corpora. GIZA++ can be used to align the tokens from any pair of sentences, which makes it a valuable tool for determining the relations between sets of tokens between lines of IGT, as well as a means to disambiguate unclear glossing. GIZA++ was also used by Xia and Lewis for their statistical method during their research for determining word order.

An important aspect of this study is using linguistic information from every line of the input IGT. GIZA++ aligns the segments of each line allowing information about one line be used on other lines of an IGT. This includes alignments of the translation to gloss line, the gloss to language line, as well as just aligning the morphemes for verbs from the language and gloss lines.

2.5 Charniak Parser

The Charniak parser (Charniak, 1997) is a statistical parser for English which utilizes a model similar to maximum-entropy. The default model is trained over the Penn Treebank (Marcus et al., 1993). The Charniak parser was utilized in this work with verbal morphology because of its use in the earlier work of Xia and Lewis described in Section 2.3. While only the part-of-speech tag information is used, utilizing the Charniak parser will allow for more information to be extracted in future experiments and provides a better point of comparison with previous works.

2.6 Morfessor

Morpho Challenge is an annual competition for automated morpheme segmentation. It began in 2005 and is still continuing presently. Morpho Challenge started as exclusively

[2] `http://old-site.clsp.jhu.edu/ws99/projects/mt/toolkit/`

working with unsupervised learning techniques for morpheme segmentation, but in later years began focusing on semi-supervised methods as well. The main baseline algorithm used by Morpho Challenge is Morfessor (Creutz & Lagus, 2006), which continues to be used as a standard in morpheme segmentation. Morfessor works at finding a balance between the compactness of the number of morphs and the compactness of the representation of the entire corpus.

Morfessor was considered as a potential means of resegmenting language information in cases where the gloss data didn't match the segmentation found in the language line. While there was potential for this technique, more work is needed to effectively combine its output with the segmentation already provided in IGT. Some techniques utilizing Morfessor are discussed in Subsection 5.2.3, although these techniques were not evaluated.

2.7 Similar Projects

2.7.1 AVENUE Project

AVENUE (Monson et al., 2008) is a project which focuses on low resource languages, specifically working on machine translation. By eliciting linguistic information from bilingual individuals who didn't have a linguistic or technical background, they were able to prototype and refine transfer rules for MT.

This study and AVENUE both focus on building tools for low-resource languages and expediting the process of building better tools. Their work focuses solely on creating rules for machine translation, while this work examines creating a grammar which only focuses on verbal morphology, but potentially can cover broader use cases. Additionally they primarily gathered information from bilingual speakers for the development of their rules, instead of already existing data from linguistically informed sources.

2.7.2 Expedition Project

Expedition was a project which also focused on working with low-density languages with the goal of creating of tools for machine translation (Oflazer et al., 2001). The goal of the

project was to ramp up the speed at which machine translation rules can be generated, specifically by providing tools for linguists to provide information about languages, and then using machine learning techniques to create and further expand on these rules.

Like AVENUE the focus of Expedition is solely on machine translation. Expedition shares some similarities with the MOM project as well, such as leveraging linguistic information, but unlike this study it elicits the information from linguists directly and requires human input before creating usable tools.

2.8 Summary

The backbone for this study, the LinGO Grammar Matrix and ODIN, was presented in this section. A look was also provided into some other contributing projects, the Charniak parser, GIZA++, and how they are used by this study to process linguistic information. Finally two older projects with similar goals were presented, and the differences in the goals and approaches used here are explained.

Chapter 3

SYSTEM DESCRIPTION

The Matrix-Odin Morphology (MOM) is a system designed to benefit from the deep linguistic information of ODIN and the Matrix customization system. It answers the questions of the Grammar Matrix customization system by extracting information from the structured data of ODIN. The MOM system uses statistical alignment to learn the connections between the translation, gloss, and language lines. These lines and their relations are the source for the answers provided to customization system questionnaire.

The focus of this study is verbal morphology. The alignment processes used are designed to help understand which sets of tokens and grams are representative of verbal aspects of the IGT. In this study when a token or token segment is marked as a *verb*, it means that the system identifies that token or segment as potentially relevant to the verbal morphology.

There are six steps which result in generating the output choices file, explained in Section 2.1.2. The choices file can be loaded without modification as answers to the Grammar Matrix customization system. The six steps for identifying the relevant tokens and creating the output files for grammar generation are:

1. Preprocessing

2. POS tagging the translation line

3. Statistical alignment of the translation line to gloss line

4. Statistical alignment of the gloss line to the language line

5. Statistical alignment of the gloss verbs to language verbs

6. Grammatical rules generation

All of these steps are explained in further detail through this section. The Example IGT 11 will be used as a running example of what the system does for each step in the entire process.

(11) *Lu-lutu-in* *ng* *lalaki* *ang* *adobo.*
Asp-cook-Acc CS man ANG adobo
'The man will cook the adobo.' (Rackowski & Richards, 2005) [tgl]

In the provided example, the system will identify the *Lu-lutu-in* as a verb, *Asp-cook-Acc* as a verbal gloss, *lutu* as the root with the meaning *cook*, *Lu* and *in* as affixes with their respective glosses, and based solely on this example no dependencies would be drawn for the relation of the two affixes besides that one appears as a prefix and the other a suffix. It should be understood that each step processes the entire set of data before going to the next step, even though only a single example will be shown going through all the steps in this paper.

3.1 Preprocessing

Preprocessing the lines of IGT prepares the lines to be interpreted by the system. Each example of IGT is categorized into groups based on the ISO annotations in ODIN. Additionally, ODIN has annotated lines for the presence of many elements such as author citations, syntactic or linguistic notes, and language names. These are removed from the data using regular expressions.

Similarly, regular expressions are used to look for non-linguistic information such as years, example numbers, page numbers, excess symbols, etc. and these are also removed. After preprocessing the IGT is less noisy and the formating of the IGT is more regularized.

3.2 POS tagging

Because POS taggers are not available for many low resource languages and are difficult to train without the appropriate training data, our process starts from the bottom of the IGT

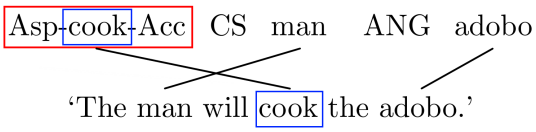

Figure 3.1: Translation to Gloss Alignment

and works up. The MOM system starts with the English translation.

The prevalence of English language processing tools makes this a straightforward task. For this experiment the set of English sentences is processed by the Charniak parser. The returned trees are examined and the tokens which correspond to verbs are marked. These marked tokens are the basis for finding the verbal tokens for the other lines.

By starting with the common element of English, the system doesn't need a POS tagger designed for each target language. This makes the MOM system able to adapt to working with any target language without further training.

In the example sentence, *The man will cook the adobo*, *cook* will be the identified verbal token. For sentences with more than one verbal token, all relevant tokens will be marked.

3.3 Translation to Gloss Alignment

The next step is to align the translation to the gloss. This step has two purposes: it identifies the verbal tokens from the gloss line, and it marks the gram of the gloss line which is the predicate of the verb. Further specifics about how the alignments in this step and the following steps are provided in Section 4.1 and Subsection 5.2.1.

In Figure 3.1, black lines mark the alignments, blue marks the tokens identified as verbs by the alignment, and the red marks tokens marked as verbs based on the smaller token marked in blue.

The MOM system aligns the list of morpheme segmented tokens from the gloss and the list of word tokens from the translation. The translation word, *cook* was marked as a verb

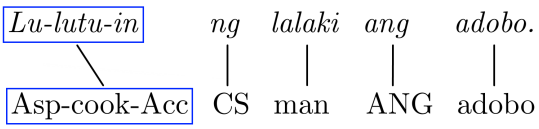

Figure 3.2: Gloss to Language Alignment

by the system in the previous step. After the alignment, the gloss *cook* and the translation *cook* are aligned. The token *cook* in the gloss line is marked as a verb, because it is aligned to another verbal token.

The system recognizes that *cook* is a part of of the larger string *Asp-cook-Acc* and will mark this larger gloss as a verb as well. Verb marks will only move from smaller token segments to larger token segments. A smaller token will give its POS tags to the larger token, but larger tokens won't pass these tags to its constituent parts. For example, *cook* is tagged a verb, and therefore so is *Asp-cook-Acc*, but *Asp-cook-Acc* does not mark its other morpheme glosses as verbs; both *Asp* and *Acc* remain unmarked. The smaller token segments marked as verbs are later used to provide the semantics as the predicates of the verb.

3.4 Gloss Sentence to Language Sentence Alignment

The purpose of the next alignment is to identify verbal tokens in the language line. This step works in the same manner as the translation to gloss alignment in the preceding section 3.3, except with different source and target lines.

GIZA++ is used to align the tokens of the gloss line with the translation line. The alignment information is used to tag the language line with the same part-of-speech as the aligned tokens in the gloss line. In the example in Figure 3.2, *Asp-cook-Acc* was already tagged a verb in the previous step. After performing the alignment with GIZA++, the alignment information returned is used to mark *Lu-lutu-in* as a verb. With the completion

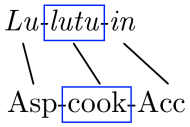

Figure 3.3: Gloss to Language Morpheme Alignment

of this step, tokens from both the language and gloss line have been marked as verbs.

3.5 Gloss to Language Morpheme Alignment

To learn which verbal morphemes of the gloss line are associated with the verbal morphemes from the language line the MOM system makes one final alignment. Unlike previous alignments, GIZA++ is provided with data where the words are treated as entire sentences, each morpheme is treated as the tokens to be aligned, and only words marked as verbs will be used for this step. The details of this type of alignment are explained in the following subsection 4.1.2.

The alignments returned by GIZA++ are used for two purposes. The first is to identify which morpheme from the language will be used as the verbal root. The second is to further determine which language morphemes will represent which syntactic and semantic features from the gloss line.

In the example, the tokens *Lu-lutu-in* and *Asp-cook-Acc* have both been identified as verbs in Section 3.3 and Section 3.4 respectively. Because of the step in Section 3.3, *cook* has been identified as a verb. Figure 3.3 shows the alignments made by the system, and the resulting marking of *lutu* as a verb. One token will be marked as a verb and that token will be analyzed as the verbal root. In the example *lutu* is marked as a verb and will be used as the verb root. The other tokens are the affixes, and their alignments tie them to the information of the gloss. This allows the system to associate the proper morphosyntactic and morphosemantic features with the proper affix orthography in the output choices file.

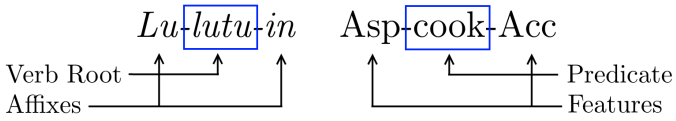

Figure 3.4: Parts Used for Rule Creation

3.6 Rule Output

The final step in the process is the creation of the choices file to be used by the Grammar Matrix customization system. The pairs of language tokens and glosses that are identified as verbs are the only strings used when answering the questionnaire. All other words from the language line are ignored.

When looking at the gloss, the string is divided into a list of grams and a predicate, and the language string is divided into the verb root and the affixes, as shown in Figure 3.4. The system will identify the verb root and the predicate based on the markings of each gram. No other grams will be marked as verbs except the verb root and the predicate, which were marked in the earlier steps explained in Section 3.5 and Section 3.3 respectively.

The system has a small set of hard coded features which recognizes the grams for many common cases such as number and plurality. Unknown grams are entered into the customization system, but each unique gram is treated as an independent feature in the questionnaire. This means that grams will be recorded as features by the grammar when appearing anywhere in the gloss. Unknown grams are recorded under the 'Other Features' heading of the customization system. Here is an example entry for an *asp* feature:

(12)
```
section=other-features
    feature1_name=asp
    feature1_type=head
       feature1_value1_name=asp_T
          feature1_value1_supertype1_name=asp
```

With a more complex means of processing grams, the Grammar Matrix customization

system allows for types which serve as values for the features to be arranged in type hierarchies. For this study, each unknown gram is left as its own independent feature. The string provided by the verb identification is used as the feature name and also as the basis for the single value. The *T* added to this string in the value name stands for *true*, because it signifies in the grammar that the feature is present.

The lexicon section of the customization questionnaire is filled out using the verbal root as the orthography and the gloss as the predicate. If there are any other grams in the gloss which align with the verb root, those grams are processed and recorded in the verb type. The lexical entry for the verb *lutu* would appear something like this:

(13)
```
section=lexicon
    verb1_valence=trans
       verb1_stem1_orth=lutu
       verb1_stem1_pred=cook
```

The signature of a verb type is determined by having the same set of features occur on the same verb as well as have the same valence. So any verbal roots with the exactly the same features specified on the root are combined into the same verb types. By grouping these verb roots, it creates rudimentary verb types which will be able to take the same affixes as the other members in its class. For example if a language had a verb type which has no features, but also a set of verbs where future tense was specified on the verb root, then these verbs would be treated as separate classes, and each verb type would inflect independently based on the example words provided in each verb type. If another set of verbs specified past tense on the verb root, these would make a third verb type. Similarly the number of features required must be the same, so if a verb was encountered where future and singular agreement was specified on the verb root, this would be an independent verb type from any verbs which only specified future tense.

Because the system is only interested in verbal morphology, every verb is marked as *transitive* for its valence value. In this example the verb root has no other grams aligned with it, so all that needs to be provided is the orthography and predicate values. Figure

2.1.2 in Section 2.1.2 provides an example of lexical entries with both features defined on the verb, as well as multiple verbs in a single verb type.

Next the affixes are processed in order of prefixes and then suffixes always moving away from the verb root. Each affix is given its own position class at first. The signature of the position class is strict at first and will only add an instance of the input if the affix the same orthography and features as the position class and both are suffixes or prefixes. If the affix matches an existing position class, then the input from the affix is added to the list of inputs of the position class. Each position class is ordered in regard to the other affixes which co-occur in at least one verb example, otherwise the ordering occurs independently of the other position classes. This independent ordering is possible because the LinGO Grammar Matrix questionnaire uses inputs as the method of establishing the order of affixes.

If a verb root is seen taking multiple prefixes and none of them occur in the same instance than each would initially be given its own position class, and each of those position classes would have the input of the single verb root. When multiple affixes occur on the same verb then the prefix closest to the verb root would have the verb root as its input. The next prefix to the left would take the prefix directly to its right as its input. Once the furthest prefix to the left is reached, then the suffixes are entered, because the input represents which morphological rule was last applied to the verb.

This process of compiling the verb types and position classes from a set of data is shown in the following pseudo code:

```
Function: build_lexicon(Array data)
Input: An array of Verb data
Output: verb_types: The list of Verb Types
Output: position_classes: The list of Position Classes
verb_types = [] ;          /* Initialize an empty list for the verb types */
position_classes = [] ;   /* Initialize an empty list for position classes */
foreach verb in data do
    /* The verb_root will contain the morpheme and gloss              */
    verb_root = verb.get_verb_root();
    if verb_root matches any verb type in verb_types then
        Add verb instance to matching verb type;
    else
        Add whole verb_root to verb_types;
    end
    verb_classes.add(verb_root);
    /* Keep the reference to verb_root for input to the next affix    */
    input = verb_root;
    prefix = verb.get_token_to_left(input);
    while prefix is defined do        /* Loop through morphemes to the left */
        if prefix matches any prefix in position_classes then
            Add input to matching prefix;
        else
            prefix.add_input(input);
            Add prefix to position_classes;
        end
        prefix = verb.get_token_to_left(input);
    end
    suffix = verb.get_token_to_right(input);
    while suffix is defined do        /* Loop through morphemes to the right */
        if suffix matches any suffix in position_classes then
            Add input to matching suffix;
        else
            suffix.add_input(input);
            Add suffix to position_classes;
        end
        input = suffix;
        suffix = verb.get_token_to_right(input);
    end
end
```

The affixes are entered into the morphology section of the questionnaire. Here is an example of two affixes entered into the morphology section of the grammar matrix ques-

tionnaire:

(14)
```
section=morphology
        verb-pc1_order=prefix
        verb-pc1_inputs=verb2
           verb-pc1_lrt1_feat1_name=asp
           verb-pc1_lrt1_feat1_value=asp_T
           verb-pc1_lrt1_feat1_head=verb
           verb-pc1_lrt1_lri1_inflecting=yes
           verb-pc1_lrt1_lri1_orth=lu
        verb-pc2_order=suffix
        verb-pc2_inputs=verb_pc1
           verb-pc2_lrt1_feat1_name=case
           verb-pc2_lrt1_feat1_value=acc
           verb-pc2_lrt1_feat1_head=obj
           verb-pc2_lrt1_lri1_inflecting=yes
           verb-pc2_lrt1_lri1_orth=in
```

The *asp* feature is not recognized by the system, so it is treated an a binary feature, while the *acc* feature is recognized as a case feature which should agree with the object of the verb. This example also shows how the inputs to a position class are output by the system. The first position class, *verb-pc1*, is a prefix which can attach to any verb which are instances of the verb types *verb2*, and *verb-pc2* will attach as a suffix to any verb where a lexical rule from *verb-pc1* had just been applied to the verb. The inputs of a position class are important for the following step.

The position classes for affixes are clustered based on which verbs and other position classes they take as inputs. The pairs are combined in order from highest percentage of shared inputs to least, and the process stops when no pairs reach a minimum input overlap percentage. The minimum input overlap percentage is computed like so:

Function: *input_overlap(pc1, pc2)*
Input: Position Class 1: pc1
Input: Position Class 2: pc2
shared_inputs = 0;
for *input in pc1.inputs* **do**
  if *input in pc2.inputs* **then**
    shared_inputs++;
  end
end
return $\frac{shared\_inputs}{length(pc1.inputs)+length(pc2.inputs)-shared\_inputs}$

By combining position classes, the resulting grammars become more general and process more verbs not seen in the training data, as well as reducing the number or choices in the output choices file. The process of combining position classes is described in Section 4.1.4.

While this set of choices is the final output of the system, the evaluation is measured by testing the grammars generated by the Grammar Matrix customization system on the basis of these choices files.

3.7 Summary

This chapter gave a high level overview of the MOM system and explained the six steps used by the system to generate the output choices files, and provides some of the details about how those steps are performed. The preprocessing of IGT data is the first step, followed by learning about the data through part of speech tagging and three alignment steps, and finally outputting the choices files.

The next chapter will look at each step in more detail. It will also provide information about the data sets used and how each step of the system was tuned to these data sets.

Chapter 4

METHODOLOGY

This chapter starts by discussing some customizations available to the system, while providing more details for the affected steps. Two data sources are used for evaluating of the overall system. The methodology developed in this study was tested using three different tests with varying datasets. The first experiment using the complete paradigm of a single French verb. The second and third experiments use IGT from ODIN for Turkish and Tagalog. This chapter presents each of these data sets in detail and the configuration for the experiments preformed; first this thesis looks at the French data and then the data from ODIN. The results of these experiments are presented in Chapter 6. This chapter also provides further implementation details throughout the chapter for each step of the MOM system process are given, in regards to the customizations available and the specific data sets used.

4.1 Customizations

The MOM system is customizable, providing multiple options for how data is processed. Certain parameters are used to leverage differing aspects of IGT. Results for parameters which are tested are discussed in Chapter 6. The main topics are bidirectionality, tokenization, boosting, and rule output parameters. The first three of these customizations affects every alignment step in the process, while the last section looks at the details of how the choices file is generated and how that output is customized.

4.1.1 Bidirectionality

Bidirectionality affects how alignments are performed. The output of GIZA++ without using bidirectional alignments is a one to many alignment process. This results in higher

recall for identifying verbs during every alignment, because more tokens can be aligned to each verb.

By using bidirectional alignments, each pair of sentences is given to GIZA++ as the source and as the target. The intersection of the two alignments is used, giving the best one-to-one match for each token. This purpose of this method is to create a more precise system, by reducing the number of alignments to each verb to the single best token.

For the tests presented here, bidirectional alignments were always used. This parameter was not evaluated but used based on the work of Xia and Lewis (2007).

4.1.2 Tokenization

How to best segment linguistic data when performing alignments is an important consideration. There are three types of segmentation used by the MOM system, which each corresponding to a level of segmentation. For any given alignment, the segmentation is a parameter for both the source and target lines, independent of which line of IGT it is. Each line of IGT lends itself to certain segmentations.

Word level segmentation divides tokens by whitespace and is the default form of segmentation for the system, because it works with any line of IGT.

Morpheme level segmentation divides tokens along the common morpheme boundary tokens such as hyphens in addition to whitespace. This is a useful segmentation when working with the finer linguistic data of the language line and the gloss line.

Gram level segmentation is when the sentences are tokenized over both symbols used for morpheme segmentation and those used for separating different aspects of the gloss line which annotate a single morpheme. For example a gloss like 'cat.pl' will be divided into two separate tokens, 'cat' and 'pl.'

Not all forms of segmentation make sense depending on the line being segmented. For example in the translation line, there isn't segmentation for morphemes. Therefore morpheme segmentation isn't a logical choice, because using word level or morpheme level segmentation will create the same sets of tokens.

Segmentation in an important consideration for the performance of boosting as explained in the following section. Further examples of how the MOM system handles segmentation are explained in Chapter 5.

4.1.3 Boosting

The MOM system treats GIZA++ as black box system and does not modify any settings used during alignment. *Boosting*[1] in the context of the MOM system is a means of improving the results from other systems by generating additional data based on well formed information.

Boosting is used as a means of improving the results returned by GIZA++. This involves processing any data before providing it to GIZA++, with the intention of finding the information which is likely more reliable and creating further data to coax the outside system to draw conclusions from the reliable data.

Alignments are improved by adding sentence pairs which consist of single words which are identified as likely matches in preprocessing. To create these one to one pairs, two styles of identification are used: looking for similar strings and looking for the same number of tokens in each sentence.

Looking for tokens which have similar strings can improve the alignment between translation lines and gloss lines. The system has three styles of matching to check the similarity of strings. The first is the least discerning, and identifies strings in the source sentence which appear anywhere in a token of the target sentence. This creates the most pairs, which is problematic with short words. For example, using this method 'at' would match words like 'cat' or 'combination.' The second method looks for words which start with the same string. This strategy attempts to match words which are the same, even in cases where the gloss is be missing any additional suffixes. For example, sets like 'run' and 'running' will

[1]The process used here differs fundamentally from the *Hypothesis Boosting* technique used in machine learning to create a strong learner out of an ensemble of weak ones, which divides data into smaller sections (Schapire, 2003). Here the data is *boosted* by creating more data from the most confident examples found using heuristic methods.

match using this method. This method works well for the translation to gloss alignment task, because English conjugates with suffixes. The final method is the most stringent and works by looking for tokens which are identical strings, but this adds very few one to one token pairs.

The second strategy is designed to improve alignments between the gloss line and language line. In well formed IGT, the number of words should be the same between the gloss and language, as well as the number of morphemes per word. However not all IGT is well formed. Well formed IGT can be used to create a one to one token pair for each token in the sentence. This data helps GIZA++ to better identify correct alignments for the ill formed IGT.

In addition to specifying the means of boosting, a *boost multiplier* can be specified. This signifies how many times additional data should be added, thereby more heavily biasing each system to rely on data which meets the boosting conditions. For example if a boost multiplier of *3* is provided while performing alignment, then any pair of tokens which matches the boosting condition with be added as three independent one to one sentences.

4.1.4 Rule Compression

One final parameter for the system controls the consolidation of the rules generated. This step modifies a choices file by combining the position classes defined in Section 2.1.2. Each position class represents a slot where an affix can occur, and by combining these position classes we are generalizing the purpose of each affix slot.

When the position classes of the customization system are combined, a compression number is used to limit this process to only position classes which share a minimum percentage of inputs. Pairs of position classes are selected one at a time to be combined, after being checked they meet the requirements. The pair must be both prefixes or affixes and must create a valid choices file after being combined. Some position classes cannot be combined and still create a valid choices file, such as in the cases where the inputs would generate a loop in the morphological rules. If the percentage input overlap is greater than

the designated minimum and the resulting choices file will still be valid, then the position classes are combined.

The process of combining position classes occurs over three steps. The first is selecting which pair of position classes shares the largest percentage of inputs. The second is merging the data of the two position classes, and the last is replacing the inputs of the other position classes with the newly constructed one. This process is presented in the following pseudo code:

```
Function: combine_position_classes(float min_overlap)
Input: min_overlap: The minimum percent overlap
/* This function just changes the state of position classes, which are
   already stored in the containing class.  This function has no output
   */
while true do
    greatest_overlap = 0;
    foreach pair of position classes, pc1 & pc2 in position_classes do
        if greatest_overlap < input_overlap(pc1, pc2) then
            greatest_overlap = input_overlap(pc1, pc2);
            pos_class1 = pc1;
            pos_class2 = pc2;
        end
    end
    if greatest_overlap < min_overlap then
        break;
    else
        /* Merge the inputs and lexical rule types from pc2 into pc1.  */
        foreach input in pos_class2.inputs do
            pos_class1.add_input(input);
        end
        foreach lexical_rule_type in pos_class2.lrts do
            pos_class1.add_lrt(lexical_rule_type);
        end
        /* Find other instances where position classes take pc2 as input
           and point them to pc1.                                       */
        foreach pc in position_classes do
            if pos_class2 in pc.inputs then
                pc.add_input(pos_class1);
                pc.remove_input(pos_class2);
            end
        end
    end
end
```

Figure 4.1.4 provides an example of combining position classes. Both verb position class one and two have only one input and it is the same verb type, verb1. The percentage overlap is calculated by taking the intersection of the inputs and dividing it by the union, as explained in section 3.6. So the percentage overlap of the inputs of position class one and two in the example is 100%. If either had a second input, the overlap would be 50%,

Before:

```
verb-pc1_order=prefix
verb-pc1_inputs=verb1
   verb-pc1_lrt1_feat1_name=case
   verb-pc1_lrt1_feat1_value=nom
   verb-pc1_lrt1_feat1_head=subj
   verb-pc1_lrt1_lri1_inflecting=yes
   verb-pc1_lrt1_lri1_orth=um
verb-pc2_order=prefix
verb-pc2_inputs=verb1
   verb-pc2_lrt1_feat1_name=rls
   verb-pc2_lrt1_feat1_value=rls_T
   verb-pc2_lrt1_feat1_head=verb
   verb-pc2_lrt1_lri1_inflecting=yes
   verb-pc2_lrt1_lri1_orth=na
verb-pc3_order=prefix
verb-pc3_inputs=verb-pc2
...
```

After:

```
verb-pc1_order=prefix
verb-pc1_inputs=verb1
   verb-pc1_lrt1_feat1_name=case
   verb-pc1_lrt1_feat1_value=nom
   verb-pc1_lrt1_feat1_head=subj
   verb-pc1_lrt1_lri1_inflecting=yes
   verb-pc1_lrt1_lri1_orth=um
   verb-pc1_lrt2_feat1_name=rls
   verb-pc1_lrt2_feat1_value=rls_T
   verb-pc1_lrt2_feat1_head=verb
   verb-pc1_lrt2_lri1_inflecting=yes
   verb-pc1_lrt2_lri1_orth=na
verb-pc3_order=prefix
verb-pc3_inputs=verb-pc1
...
```

Figure 4.1: Two Choices File Fragments Before and After Combining Position Classes

and if they both had a different second input, then the overlap percentage would be 33%, because they would only share one input out of three total.

Once the position classes are selected to be merged, the lexical rule types are compared and those with matching feature values are combined. In our example the lexical rule types do not have the same features, so they are left as different lexical rule types. If they did have the same features then the two lexical rule instance lines would be combined under the same lexical rule type.

Finally, the inputs are adjusted. The inputs of the two merged position classes are combined. In all cases, the number of inputs for the new class will be greater or equal then the number of inputs for both of the previous position classes. Also the inputs of all other position classes are examined, and any which point to one of the merged position classes is

Verb Form	Number of examples
Present	2420
Past Perfect	2420
Past Imperfect	2420
Future	2420
Conditional	2420
Present Subjunctive	1430
Past Subjunctive	1430
Imperative	474
Present Participle	110
Past Participle	4
Infinitive	110

Table 4.1: Distribution of Verb Forms of the French Data.

linked to the new combined position class. In the example we see that a third position class which originally has verb-pc2 as an input, now has the new verb-pc1.

The lower the threshold for minimum input overlap, the more the system generalizes the rules from the training data, which causes an increase to the number of examples it can parse, but also can lead to an increase in overgeneration. These gains are caused by the increase in number of paths by added new inputs and allowing other position classes from using either as an input. This step also reduces the number of rules defined in the choices file.

4.2 Regularized French Data

4.2.1 French Data

The French experiment data is a compilation of all the forms of the verb *faire* written phonetically in IPA and regularized to include liaison consonants in all cases, and treats "clitics" as affixes (cf. Miller & Sag, 1997). This data set includes four indicative forms, two subjunctive forms, as well as imperative, infinitive, conditional, present participle, and past

participle forms; each variation is given with the all the variations of the allowed pronominal clitics. The distribution of verb forms is shown in Table 4.1. With all these variations, the data set of grows to include 15,658 forms of the single verb. All the forms of the verb were generated automatically using a Perl script written by Olivier Bonami. In order to be consistent with the data in ODIN, presented in Subsection 4.3.1, a mock translation line with the word *do* was added to each pair of gloss and phonetic form. Below is an example of one such constructed IPA entry:

(15) *ʒə-fɛ-z*
NOM.1SG-do.PRS-1SG
"do" [fra in IPA]

The amount of data used for each test was varied to simulate different amounts of IGT available. The data was randomly divided into 1000 non-overlapping sets. To evaluate the performance of the system with smaller and smaller amounts of data, they were recombined into groups of diminishing sizes covering steps of 10% down to 10%, then steps of 1% and finally steps of 0.1% of the total amount of data.

4.2.2 Gold Standard

The over fifteen thousand forms of *faire* which represent the entire set of grammatical forms are used as the test case. To provide a point of evaluation for overgeneration, 12,208 incorrect forms were created. The techniques used in the creation of the ungrammatical forms include adding non-morphemes, and rearranging and duplicating existing morphemes. Each incorrect form is known to be incorrect, because it is checked to not be in the exhaustive set of correct forms.

These forms are used to create the gold standard by formatting them as an [incr tsdb()] testsuite (Oepen, 2001). After a grammar is generated from the choices file, it is loaded into the LKB, and the LKB parses each item of the gold standard testsuite using the input grammar. The coverage of the grammar is calculated by taking the percentage of

grammatical forms parsed, and overgeneration is calculated by finding the percentage of incorrect forms parsed. The evaluation metrics are further explained in Chapter 6.

The most important part of each item in a the testsuite is the language lines and the grammaticality judgment. Further information is also provided for each line, such as the gloss for grammatical items. Since these were machine generated each is marked false for vetting, and no translation is given for any item. Example 16 shows two testsuite items, the first is grammatical and the second is ungrammatical.

(16) Source: a:1
Vetted: f
Judgment: g
phenomena: verb morphology
n1 ʒəfɛz n1
n1 ʒə-fɛ-z n1
n1 NOM.1SG-do.PRS-1SG n1
No translation given.

Source: author
Vetted: f
Judgment: u
Phenomena:
n1 ʒənəfi n1
n1 ʒə-nə-fi n1
No gloss given
No translation given.

4.2.3 Process

The MOM system takes in the plain text versions of the mock IGT as the input. After processing the data, a choices file is output, and each choices file is evaluated after using the Grammar Matrix customization system to compile the choices file into a grammar fragment which is used to parse the gold standard testsuite.

Due to the standardization of the French data, the primary steps to align the tokens for each section of the IGT performed perfectly. When the amount of data was reduced drastically, boosting as described in 4.1.3 was required to maintain correct alignments. The steps here are designed to use the same means of processing data as those used for ODIN,

because the French data acted as a proof of concept it made sense to keep the tests as similar as possible.

During the first stage, because the same mock translation was added for each instance of IGT, "do" was always correctly marked as a verb after running the sentences through the Charniak parser. Exact string boosting as explained in Section 4.1.3 was used during the alignment of the translation lines to the gloss lines. For the next two alignments, which determine the alignments of gloss words to language words and verbal glosses to verbal morphemes, exact number of tokens boosting was used to guarantee perfect performance of the alignments. While for most data sets the boosting was unnecessary, when reducing the data to groups less than a percent of the total data, in rare instances GIZA++ would misalign segmentations, because common grams could occur on the gloss line as often as the "do" token.

The gloss/verb pairs were then turned into choices and rules to be compiled into grammars. Each test underwent multiple levels of compression of position classes, in steps of 10% from no compression to combining any position class with a single shared input.

The results from each test are reported in Section 6.1.

4.3 ODIN

4.3.1 ODIN Data

The rest of the experiments used the data taken from real linguistic articles compiled by ODIN (Lewis, 2006). The database of ODIN used for this study contains 158,007 examples of IGT ranging over 1495 languages.

The amount of data varies depending on the language. Table 4.2 shows the distribution of the IGT among the languages. Tagalog and Turkish were selected for their interesting verbal morphology and the amounts of data available. Tagalog has 1,039 total examples of IGT in ODIN, and Turkish has 2,617 total examples.

Range of IGT Instances	Number of Languages	Total Instances of IGT
10,000+	1	10,814
1,000-9,999	31	81,218
100-999	141	47,306
10-99	460	15,650
1-9	863	3,019

Table 4.2: Number of Languages Classified by Amount of IGT

4.3.2 Gold Standard

For both languages the data was divided into two categories: the training data (90% of examples) and the testing data (10% of examples). The testing data was removed from the process entirely and reserved for evaluation only. The Tagalog gold standard contains data from 104 randomly selected examples of IGT for the testing data, and the Turkish gold standard contains data from 262 randomly selected examples of IGT. The gold standard was then developed by hand from the testing data. It contains the verb tokens found in the language lines for each example of IGT, which is then used to create an [incr tsdb()] testsuite.

A portion of training data equal in size to the testing data was also hand processed to create the development set. Like the training set, the development set for Tagalog also contains 104 randomly selected examples of IGT and the Turkish data contains tokens from 262 randomly selected examples of IGT. Because the steps taken by the MOM system do not use the development set as training for any supervised learning, the development set allows for evaluation of each step in the process and the evaluation of the multiple modes for improving performance.

The gold data for the development set contains the unique IGT id followed by any verbal tokens from all the lines of IGT. It includes the verb forms found in the language lines, as well as their glosses, and the single word which best translates the verb in the translation

line. These forms are always kept as a list of three tokens.

1. The translation of the verb
2. The gloss of the verb
3. The verb form

Because all three tokens are required by the system, incomplete IGT examples are not included in the gold standard. Example 17 shows an example of such gold standards entries. The numbers are the IGT ids and each following line has the three tokens which represent verbs for each line of IGT. Additionally any verbs found in the translation will not be marked as such if they aren't a translation of a verb in the target language, and similarly some non-verbs are marked in the translation line if they are the best translation for a verb in the target language.

(17)
```
1184 7197 7202:
ordered PERF-order-IO in-utus-an
kissed NONVOL-kiss-IO ma-halik-an

1183 5572 5574:
bought PERF.AV-buy b-um-ili
```

In the case there was an error in the raw text data, such as additional or missing whitespace due to how the data was scraped, the original correct form from the source was recorded in the gold standard. While this makes impossible for the current system to match these tokens, it allows for a gold standard set which can continue to be used in future experiments. Additionally it keeps a focus on developing high precision grammars.

4.3.3 Process

The system takes in the raw text ODIN IGT as its input. The output for each configuration is a choices file which can be used by the Grammar Matrix customization system to create a working HPSG grammar.

Like in the French data, the marking of verbs in the translation has no additional parameters. Every alignment step will use bidirectional alignments, to help filter results to the tokens of interest.

When aligning the translation to the gloss line, different segmentations are used for each line. As the translation line has no morpheme segmentation marked, the string is divided at the word level, which segments along whitespace. For the gloss line, separating along the morpheme boundaries or dividing each gram into its own token each provides a possible advantage. For the experiments, glosses were either segmented at the morpheme boundaries or gram boundaries, allowing the predicates, syntactic, and semantic features to all be aligned as separate tokens.

This choice goes hand in hand with another option. Because the gloss line and translation line of the IGT of interest are both written in English, the amount of language dependent differences for these two lines can be quite minimal. One method to try and improve performance is to train the alignment model on the entire data set of ODIN instead of merely over the IGT training set for the target language. Xia and Lewis (2007) were able to use the same technique to improve alignments in their work. This is related to how segmentation is performed, because when aligning segmentations of each individual gram and predicate, it increases the amount of language independence when performing the alignments. While it's possible that by aligning tokens segmented at the morpheme boundaries will allow for GIZA++ to learn more of the nuances of the language and allow for easier alignments of specific conjugations, alignments using gram boundaries outperformed morpheme segmentations. Finally boosting can be used during this step as a means of improving the alignments. The benefits of adding matched tokens to the data is presented in Chapter 6.

The next alignment is the alignment of gloss words to the language words for purpose of identifying verbs. Alignment boosting is used to add data for sentence pairs which are well formed, and the evaluation of gain from this boosting can be found in the Chapter 6.

Because the system will utilize alignment boosting for this step, there are only two

logical choices for segmentation. Both segmenting along word boundaries and morpheme boundaries are tested by the system. In the best formed IGT, segmenting along morpheme boundaries provides the system with the best opportunities to learn the patterns, but in many examples, not all morphemes will be segmented exactly the same in both lines. This is especially true for more commonly used languages, or in words in the language line which were not of interest to the source paper. These typically result in verbs in the choices file which will be overspecified and won't take any affixes or underspecified and will take incorrect affixes.

For the final alignment step of aligning the verbal morphemes, as in previous steps, the alignments are performed as bidirectional. The segmentation is at the morpheme level for all the sets of words. The only parameter evaluated for this step is the use of boosting to help improve alignment by adding data based on word pairs with matching amounts of segmentation for the gloss and language strings.

The lexical rules are generated and then output uncompressed and after four additional levels of compression, which combine position classes sharing 100%, 80%, 60%, 40%, and 20% of inputs respectively.

4.4 Summary

This chapter presents the ways the MOM system can be customized, such as bidirectionality, tokenization, and boosting, as well as some of the details of the steps affected by each customization. It then examines the three experiments preformed in this study, describing and contrasting the data sets used as well as how the system was tuned for each data set. The next chapter will go further into how the data is stored and analyzed in the system.

Chapter 5

INTERNAL REPRESENTATION

This chapter goes into the specifics of the internal storage and interfaces used for processing the IGT data. Although the information in this section is not necessary for understanding the results presented in Chapter 6, it provides the reader with a better understanding of how data is handled internally and the potential for future modifications.

The MOM system provides a simple high level module in python for users to easily work with a large amount of IGT and easily configure experiments for testing. The system provides methods for reading ODIN formatted files, processing and manipulating IGT, and outputting the information in the forms of a choices file. The modules which make up the MOM system include an IGT module, alignment module, parser module, morpheme analyzer module, and a driver module.

5.1 IGT Module

The IGT module is at the core of the MOM system. It is divided into a hierarchy of classes for storing and manipulating IGT, each containing an array of the class beneath it. The classes from highest level to lowest are the IGT_Library, IGT_Book, IGT_Word, IGT_Morpheme, and at the lowest level the IGT_Feature[1].

5.1.1 IGT Database

IGT_Library is the main class used by the entire IGT module. It provides access to the functionality of all the lower classes in the hierarchy. The major functions of the IGT_Library are reading in ODIN formatted files, storing and reading processed IGT information, print-

[1]The name *feature* is used for this object because they are most commonly represent what will become features in the choices files, but this class is used to represent every gram of the gloss line.

ing a choices file, and dividing the data into training and testing groups. The IGT_Library also provides the functions used to interface with the other modules used for performing alignment, splitting morphemes, and parsing translation lines.

The IGT_Library contains an IGT_Book for each different language. Specifically an IGT_Book will contain all the examples of IGT which have the same iso language code. The IGT_Book objects aren't accessed directly, but instead by making function calls from the IGT_Library.

5.1.2 IGT Representation

An instance of the IGT class is created for each example of IGT. It stores annotation information such as language and document id. After processing an IGT object the IGT_Line objects will be clearly divided into the language, gloss, and translation line objects. The original lines can maintained during processing steps, but by default are deleted to reduce memory use.

The IGT_Line class is the base class for the Translation_Line, Gloss_Line, and Language_Line classes. IGT_Line groups together all the tokens found on an individual line, and stores any annotations of the ODIN data. Each specialized class provides additional methods for working with the specifics of each line, such as removing non-linguistic tokens, data, or symbols. IGT_Line objects are the most commonly used object when performing alignment, as most alignments are looking to align tokens or grams across the different lines of the IGT.

(18) *Kitab-i* *adam* *oku-du*
Book-ACC man.NOM read-TENSE
'The man read the book.' (Bozsahin, 2002) [tur]

In the Example 18: *kitab-i adam okudu* is used to create the Language_Line object, *Book-ACC man.NOM read-TENSE* for the Gloss_Line object, and the Translation_Line object is created from *The man read the book.* Together these three lines comprise the IGT object.

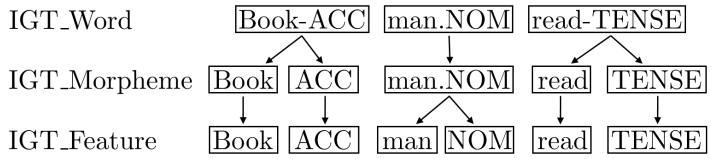

Figure 5.1: Tokenization of the Gloss Line

Once each line is tokenized it generates sets of IGT_Feature, IGT_Morpheme, and IGT_Word objects. These three classes inherits from a base class known as the IGT_Element. The IGT_Element provides the basic functionality which is needed by each part of an IGT object, such as the storing and retrieval of ids, alignments, and part of speech information, and parent object information. The alignments and part of speech tagging are all handled through the system by high level functions, and they can be checked or modified with the appropriate function calls. Figure 5.1 shows the different tokenizations levels of IGT_Elements for the gloss from Example 18.

IGT_Word objects store the tokens of each line segmented by whitespace. It maintains unique ids and tags for each token in all three lines. The IGT_Word also can act as a group of alignable objects, such as the alignments used in Section 3.5.

IGT_Morpheme stores the tag information and alignments for each morpheme. For the gloss line, the IGT_Morpheme class represents groups of grams and lemmas which are associated with only one morpheme. Since the translation line lacks morpheme segmentation, the morpheme level will simply have one IGT_Morpheme instance per IGT_Word instance.

IGT_Features are the lowest level object used by the system. In the gloss line, they represent the grams and lemmas. The grams eventually become features in the Grammar Matrix customization system. The IGT_Feature is primarily used with gloss lines, but every IGT_Morpheme is divided into IGT_Feature objects. In the case of language and translation lines, the result is one IGT_Feature per IGT_Morpheme in those lines. This

allows the system to maintain general functions for processing any line of IGT and expect the same functionality.

5.2 Related Modules

Two modules are used for processing of the data in the IGT module. The GIZA module interfaces with GIZA++ and performs alignments. The parser module provides a method for tagging tokens with part of speech information. Both aren't called from the main driver directly, but interface with the IGT module by using IGT function calls.

5.2.1 GIZA Module

Alignments are made available by the IGT_Library, but performed by the GIZA Module. The IGT_Feature, IGT_Morpheme, and IGT_Word classes all inherit from the GIZA Alignable class, which allows them to be aligned by the module. IGT_Word and IGT_Line classes inherit from the Alignable_Group class, which provides the infrastructure for encapsulating tokens to be aligned. Alignable_Groups are also created for use with the boosting techniques explained in Section 4.1.3.

The GIZA module performs alignments for any two lists of Alignables_Group objects, with a parameter for specifying bidirectionality. As the GIZA module is separate from the IGT module any processing done to generate additional data and extracting the correct level of segmentation is handled before the data is passed to the GIZA module.

Additional processing is sometimes required for extremely long sets of tokens or those where one sentence has many more tokens than the other. Due to the fertility rate used by GIZA++, any sentence pairs where either sentence contains nine times more tokens than the other must be removed, as well as sentences which contain more than one hundred tokens. In these cases, the pairs are removed, but the word to word probabilities created by GIZA++ during normal alignment are used to calculate the most likely one to one alignments. This fall back method will always create one to one alignments even if the bidirectionality option is set to False. Once the alignments are completed, the Alignable objects have stored the

set of other object to which they are aligned.

5.2.2 Parse Module

Another module used by the IGT_Library is the Parse module. The Parse module simply calls the Charniak parser, and reads the output parse file. It takes in a list of sentences and sentence ids, and then creates a dictionary of ids and parse trees. The IGT_Library uses this module for the identification of POS tags for tokens on the translation line. Also the Parse module provides a framework which in the future could be used to interface with other parsers or part of speech taggers.

5.2.3 Morph Analyzer Module

The final module is the Morph_Analyzer, it is the prototype module for splitting morphemes. The benefits of this system were not evaluated, and this prototype module was not used in results presented in this paper, although may be used in future works.

Like the GIZA module, most of the processing is performed by the IGT_Library before the data is given to the Morph_Analyzer module. Once a word list has been given to the module, it prints the list without any morpheme segmentation and runs Morfessor. The results are used to add additional segmentation to the existing segmentation. These new segmentations are then read by the IGT_Library to re-segment each word. Because the segmentation creates new morpheme and feature objects, any tag information and alignments previously on those objects are lost. Therefore re-segmentation should be completed before alignments.

5.3 Summary

This chapter presented the classes and modules which comprise and are used by the MOM system. These classes are arranged in a basic hierarchy and allow for a high level of customization when reading and processing data.

Chapter 6

RESULTS

This chapter presents the evaluation of the MOM process for each step and once more after system has generated the final output, using appropriate metrics for each language of each dataset. Precision, recall and f-score are used to score the classification task of identifying aligned tokens with a verb in the language line. The final evaluation consists of two metrics, coverage and overgeneration. The coverage of the grammar measures the percent of morphologically correct verbal forms for which the grammar returns a parse. The overgeneration of the grammar measures the percent of the morphologically incorrect verbal forms for which the grammar incorrectly returns a parse. This chapter also presents information about the output number of choices, verb types, and position classes.

This chapter presents the coverage and the number of generated choices, verb types, and position classes for all three experiments. When examining the results of the French data experiment, no results for the verb classification task are presented, because the system performed perfectly. The system also evaluated the overgeneration of the French grammars uses the generated dataset of incorrect forms. This chapter provides the results for each stage of the verb identification task when using the ODIN data. The system did not evaluate the overgeneration of the grammars from the ODIN data, due to a lack of gold standard negative examples.

6.1 Regularized French Data

This section examines the coverage and overgeneration of the grammars generated when working with the French data described in Section 4.2. Training used different amounts of data, but the testing is always over the entire set of grammatical and ungrammatical forms.

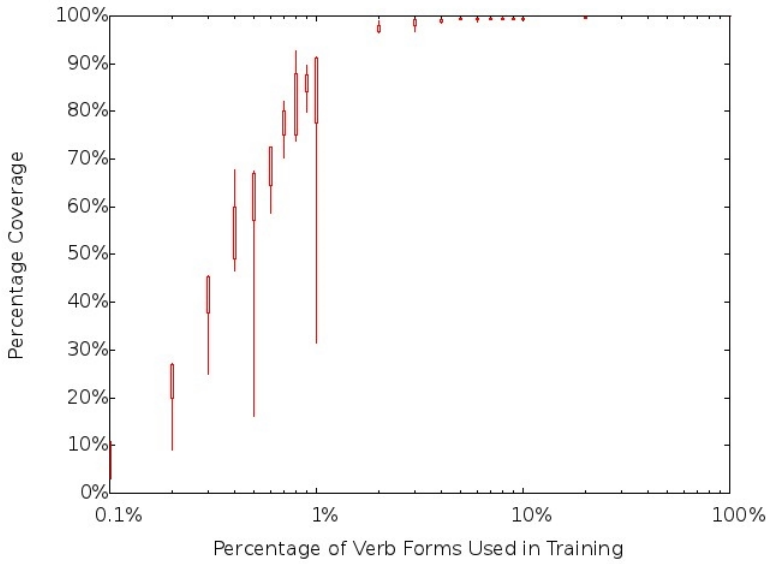

Figure 6.1: Coverage of Verb Forms by Amount of Training Data, Log Scale

6.1.1 Coverage

Tests were performed over steps of 10% of the entire set of data, down to 10%. The data was further divided sets from 1% to 10% by steps of 1%. Even smaller splits of data were taken below 1% at increments of 0.1% There was one grammar generated for each data set ranging from 30-100% of the data, five non-overlapping grammars where generated which used 20% of the total data, and ten grammars were generated for all smaller division of data. The results for the coverage of the generated grammars before reducing position classes can be seen in Figure 6.1. In the smaller sets of data, outliers did occur when a segment of data wouldn't have any examples of a morphological feature. For example, with only 156 examples it is possible to miss an entire feature, such as negation, which would contribute to an inability to parse a large portion of the correct grammatical forms. But even so, with only 1% of the total data, the ten grammars average around 80% coverage. This shows that

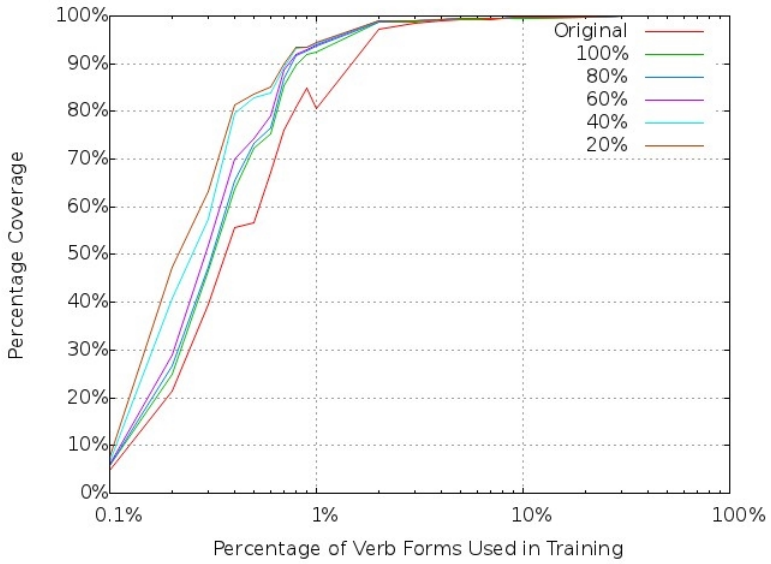

Figure 6.2: Average Coverage of Verb Forms by Minimum Input Overlap, Log Scale

the system is very capable of generalizing to unseen forms. This capability stems from the model of the morphology as a combinatoric system used in the choices files, which is the same theoretical model used by the grammars generated from these choices files.

These results show the coverage of the grammar before any rule compression was used, so each lexical rule is assigned a position class. The coverage can be increased by combining position classes, using the methods explained in Section 4.1.4. Each choices file was tested after different levels of overlapping input percentages. The average improvement to coverage by minimum compression limit can be seen in Figure 6.2. Grammars which neither performed poorly nor exceptionally well showed the greatest benefit from the generalization of rules.

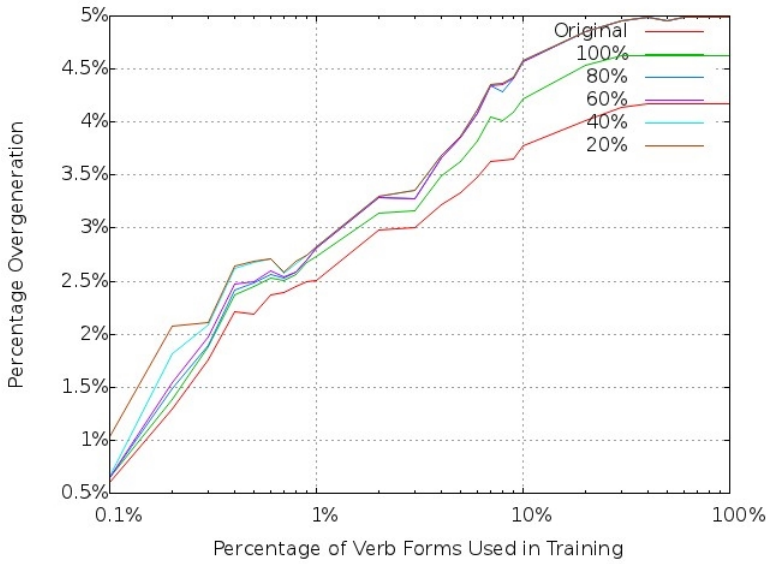

Figure 6.3: Average Overgeneration of Verb Forms by Minimum Input Overlap, Log Scale

6.1.2 Overgeneration

As position classes are combined and more forms can be parsed by the grammars, in addition to improving coverage, more incorrect forms will also be licensed and the overgeneration will increase. The grammars which already parsed the largest numbers of correct forms showed a larger absolute increase in accepted ungrammatical forms after compression. The overgeneration of each grammar is presented in Figure 6.3. The overgeneration stayed relatively low with the most incorrect forms parsed only representing 5% of the total incorrect forms.

6.1.3 Choices

The output of each run is a choices files like those explained in Section 2.1.2. The number of choices provides a general metric for the amount of data contained in each choices file.

The number of verb types and position classes provides some information about what sort of data has been generated. Unlike coverage or overgeneration, this isn't a metric which is scored, but just statistical information about the generated output.

Table 6.1 shows the average counts for the choices, verb types, and position classes. As less data is used there is a decrease in the number of choices overall, as well as verb and position classes. Like the coverage, the number of choices drops off beginning around when only 10% of the data is used, and the drop becomes more dramatic after reduced to less than 1%.

Compression will also effect the number of Position Classes, and by extension the number of choices overall. Compression allows the choices files to generate grammars with more generalized rules, but combining the lexical rule instances of a position class together. The end result being grammars which are defined by fewer choices but have increased coverage. Table 6.2 shows the average number of position classes for all the choices files generated. The first line which uses all 100% of the data, provides a point of comparison for the rest of the table, and demonstrates that only 17 position classes could be used to still have the same coverage as 70 position classes.

6.2 ODIN

The results presented here are compared against a hand evaluated gold standard, as explained in Section 4.3.2. Scores are presented for each step following the best performing configurations for the previous steps.

6.2.1 Verb Identification Task

Because the verb identification through alignment is primarily an unsupervised learning task, the development set was used for evaluation of these intermediate steps. This allows for a greater understanding of the behavior of the system, the gains made by certain techniques, as well as some insight to where error is introduced. The verb identification task uses a random baseline for comparison. For each instance of IGT a random token was selected as

Percentage of IGT	Choices	Verb Types	Position Classes
100%	938.0	21.0	70.0
90%	923.0	20.0	70.0
80%	907.0	19.0	69.0
70%	911.0	19.0	70.0
60%	926.0	20.0	70.0
50%	938.0	21.0	70.0
40%	911.0	19.0	70.0
30%	911.0	19.0	70.0
20%	886.0	17.8	68.8
10%	875.0	17.2	67.8
9%	869.5	17.1	67.4
8%	872.1	17.1	67.6
7%	868.0	17.0	66.9
6%	866.1	16.9	67.1
5%	853.7	17.2	65.2
4%	845.2	16.6	64.6
3%	832.8	16.5	63.0
2%	819.0	16.1	61.3
1%	780.7	15.0	58.1
0.9%	762.8	14.3	57.3
0.8%	761.3	14.3	56.9
0.7%	753.1	14.1	56.2
0.6%	753.5	14.7	55.9
0.5%	718.1	13.0	54.7
0.4%	716.9	13.1	54.7
0.3%	638.5	10.9	48.4
0.2%	566.4	9.4	43.4
0.1%	397.0	7.8	29.0

Table 6.1: Average Counts for the Choices, Verb Types, and Position Classes

Percentage IGT \ Compression	None	1	0.8	0.6	0.4	0.2
100%	70.0	24.0	17.0	17.0	17.0	17.0
90%	70.0	24.0	17.0	17.0	17.0	17.0
80%	69.0	23.0	16.0	16.0	16.0	16.0
70%	70.0	24.0	17.0	17.0	17.0	17.0
60%	70.0	24.0	17.0	17.0	17.0	17.0
50%	70.0	24.0	17.0	17.0	17.0	17.0
40%	70.0	24.0	17.0	17.0	17.0	17.0
30%	70.0	25.0	16.0	16.0	16.0	16.0
20%	68.8	23.6	15.8	15.8	15.8	15.8
10%	67.8	24.8	15.3	15.3	14.9	14.9
9%	67.4	23.6	15.0	14.8	14.4	14.4
8%	67.6	24.6	15.8	15.7	15.2	15.2
7%	66.9	24.0	14.9	14.9	14.4	14.4
6%	67.1	23.7	15.2	15.2	14.2	14.2
5%	65.2	23.0	14.8	14.8	14.1	14.0
4%	64.6	22.6	15.3	15.2	14.1	14.1
3%	63.0	24.5	16.5	16.5	14.3	14.3
2%	61.3	23.8	15.9	15.5	14.2	14.0
1%	58.1	25.8	16.8	16.2	14.6	13.8
0.9%	57.3	25.5	17.4	16.9	14.5	14.2
0.8%	56.9	27.4	19.5	18.5	13.8	13.4
0.7%	56.2	27.2	18.8	17.4	14.3	13.4
0.6%	55.9	29.7	23.6	21.3	15.1	14.2
0.5%	54.7	30.8	24.9	23.5	13.7	12.9
0.4%	54.7	32.3	26.3	23.6	14.5	13.1
0.3%	48.4	29.8	26.7	23.9	15.7	13.0
0.2%	43.4	29.6	27.7	26.6	14.7	11.5
0.1%	29.0	21.6	21.2	21.0	18.8	11.6

Table 6.2: Average Counts for Position Classes Based on Amount of Data Used and the Level of Compression.

Language	precision	recall	f-score
Tagalog	0.828	0.923	0.873
Tagalog Baseline	0.143	0.145	0.144
Turkish	0.789	0.882	0.833
Turkish Baseline	0.168	0.162	0.165

Table 6.3: Verb Identification in Translation Lines

the *verb* of the IGT. Selecting one verb per IGT is close to the ratio of verbs per IGT, which is 0.9811 for Tagalog and 1.034 for Turkish. In general the coverage should be considered the final evaluation for each grammar.

The first evaluation is the identification of verbal strings in the translation line. The system is trying to identify strings which are specifically translation of verbs in the language line. This is different than identifying the verbs found in the translation line, because not all English verbs will translate into verbs in the target language and visa versa. Because no other parameters were tested during this step, this evaluation provides a point of comparison for the results of identifying verbs after the alignment process, as the system cannot identify more verbs than those found in the previous step, which leads to a degradation of recall through the process.

The results are presented in Table 6.3. Tagalog has higher recall than precision with 92.3% recall and 82.8% precision. Similarly Turkish has a higher recall at 88.2% and a precision of 78.9%, which both start off lower than Tagalog. For this first step, maintaining the higher recall is more important than keeping a high precision. The recall can only decrease with each step, while the precision can improve when false positives are removed though the alignment processes.

The next evaluation is of the alignment to the gloss line. Three different techniques were used as ways of improving the performance of the alignments.

The identification of verbs on the gloss line is seen in Table 6.4. The system performed

Language	Source of Sentences	Token Segmentation	Precision	Recall	F-Score
Tagalog	All of ODIN	Gram	0.922	**0.732**	**0.816**
Tagalog	All of ODIN	Morpheme	**0.929**	0.536	0.680
Tagalog	Tagalog Only	Gram	0.702	0.340	0.458
Tagalog	Tagalog Only	Morpheme	0.632	0.247	0.356
Tagalog Baseline	N/A	N/A	0.158	0.172	0.165
Turkish	All of ODIN	Gram	**0.776**	**0.697**	**0.734**
Turkish	All of ODIN	Morpheme	0.756	0.533	0.625
Turkish	Turkish Only	Gram	0.732	0.549	0.628
Turkish	Turkish Only	Morpheme	0.714	0.389	0.504
Turkish Baseline	N/A	N/A	0.244	0.260	0.252

Table 6.4: Verb Identification in Gloss Lines

as expected with the highest performance occurring when using all the available sentences in ODIN and segmenting the tokens into grams for both languages. Tagalog best performance showed a 73.2% recall and 92.2% precision during this step, and Turkish had a 69.7% and 77.6% precision. The effect of using all of ODIN for this alignment is quite drastic as it increased the recall of both languages about 15-27%, absolute. The increase to precision was much more drastic using all of ODIN in the Tagalog with an absolute increase of around 20-30%, while the improvement in Turkish was only a few percentage points. This difference between languages could be due to the amount of data for each language, as well as the consistency of the glossing.

In the examples of IGT, the words used in the translation line are often the same as those in the gloss line. So the boosting technique explained in Section 4.1.3 can be used to add example sentences of single tokens, specifically, if the gloss matches the start of the token in the translation. With this technique the results improved more is the cases with less data available. The improvement when using all of ODIN for alignment is significantly less, and really only seen in the recall of the system. The results of boosting for the highest scoring tests can be seen in Table 6.5.

Language	Source	Segmentation	Boosting	Precision	Recall	F-Score
Tagalog	All of ODIN	Gram	True	**0.913**	**0.753**	**0.829**
Tagalog	Tagalog Only	Gram	True	0.871	0.629	0.730
Turkish	All of ODIN	Gram	True	**0.761**	**0.705**	**0.732**
Turkish	Turkish Only	Gram	True	0.750	0.615	0.676

Table 6.5: Verb Identification with Boosting in Gloss Lines

For both languages boosting led to an increase in recall and a decrease in precision. The precision in Tagalog decreased an absolute 1.6% to 91.3% overall, while the recall increased an absolute 1.9% to 75.3%. In Turkish the effects were similar with an decrease in precision of an absolute 1.5% to 76.1% and an increase in recall of an absolute 0.8% to 70.5%. This resulted in a 0.2% decrease in f-score for the Turkish. In this case, boosting worked as a means of generating more alignments by Giza++ which allowed it to find more of the correct strings, but also maintained more of the error from the previous steps. Because maintaining a higher recall is important for these early steps, despite the slight drop in f-score, the system performed better down the pipeline with boosting.

The overall evaluation is looking at the identification of verbs in the language line. All the results for this step are after using Gram segmentation, IGT data from all sources, and boosting for the translation to gloss alignment. Table 6.6 shows the results of the different segmentations with and without boosting during the gloss to language alignment. Boosting during this step leads to an increase in both precision and recall for both languages. The best performances for recall for both languages occurred using word segmentation, with Tagalog at 71.7% recall and Turkish with 62.3% recall. The better precision for both languages was using morpheme level segmentation. Tagalog has 84.6% precision and Turkish has 76.1% precision. These differences are because using a finer grain segmentation makes boosting only improve the best formed IGT, which increases the precision, but limits the cases where verbs are not found with such tokenization.

Language	Segmentation	Boosting	Precision	Recall	F-Score
Tagalog	Word	False	0.822	0.707	0.760
Tagalog	Morpheme	False	0.803	0.576	0.671
Tagalog	Word	True	0.825	**0.717**	**0.767**
Tagalog	Morpheme	True	**0.846**	0.598	0.700
Tagalog Baseline	N/A	N/A	0.138	0.160	0.148
Turkish	Word	False	0.776	0.615	0.686
Turkish	Morpheme	False	0.773	0.607	0.680
Turkish	Word	True	0.766	**0.623**	**0.688**
Turkish	Morpheme	True	**0.761**	0.619	0.683
Turkish Baseline	N/A	N/A	0.240	0.261	0.250

Table 6.6: Verb Identification in Language Lines

The system preformed best with both languages when segmented along word boundaries. The effects of boosting were mostly positive, although did decrease the precision for the Turkish grammars. For the results presented in the following section the test grammars were generated using the word level segmentation and boosting.

6.2.2 Coverage

As a comprehensive form of evaluation, the coverage of the gold testsuites, that were generated from the held out IGT, is presented for each generated test grammar. The coverage of a naive baseline is also provided as a means of comparison.

The naive baseline simply takes the set of strings identified as verbs by the system and compares the testsuite items to this list, ignoring morpheme segmentation. This baseline is naive both in its approach to the narrow task of recognizing correct verbal forms and in the actual output it provides. The naive baseline does not create a grammar, and therefore cannot be built out to greater coverage, nor does it provide any analysis of the examples it recognizes.

One final alignment is performed before the grammar can be generated. This aligns the

Language	Boosting	Coverage	Baseline Coverage
Tagalog	False	0.784	0.803
Tagalog	True	**0.794**	**0.803**
Turkish	False	0.623	0.459
Turkish	True	**0.630**	**0.459**

Table 6.7: Coverage of Test Data with Boosting in Morpheme Alignment

morphemes of the verbs in the language line with the grams from the verbs of the gloss line. This provides two grammars per language using alignment boosting for the final alignment. Again boosting increases the coverage over the test data as shown in Table 6.7. In the case of Tagalog, the baseline slightly outperformed the system before rule compression. While the generated grammar did parse a few forms unrecognized by the baseline, the number of relevant forms which weren't used to generate rules due to poor segmentation was higher.

In each case there are a few forms the naive baseline will match which sometimes do not make it into the final grammar. This happens when the author's gloss segmentation doesn't match the segmentation of the language line. So while the string is identified as a verb and matches an item in the testsuite, the output grammar isn't able to output proper rules to cover that case. This accounted for five (4.9%) forms which the baseline correctly identified which the system did not, and all five of the cases were attributed to improper segmentation in the original IGT. The system was able to generalize to cover four (3.9%) cases which the baseline failed to identify.

It should also be understood that the naive baseline lacks the complexity which is provided by using a grammar fragment, as the baseline merely compares the strings. The HPSG fragment keeps track of much more complex information which is valuable for future grammar development. The information provided to the Grammar Matrix customization system is use to output valid Type Description Language (tdl) files. For example here are some fragments of automatically generated inflectional rules in tdl from a Tagalog grammar:

```
(19)  verb-pc21-lex-rule := add-only-no-ccont-rule & infl-lex-rule & verb-pc20-rule-dtr &
         [  DTR verb-pc21-rule-dtr,
            SYNSEM.LOCAL.CAT.VAL.SUBJ.FIRST.LOCAL.CONT.HOOK.INDEX.PNG [ PER 1st,
                                                                        NUM singular ] ]

      verb-pc21-suffix :=
      %suffix (* ko)
      verb-pc21-lex-rule.
```

Using many such rules derivation trees are created for each parse, where each node in the tree will provide an additional affix to the verb form. The final results of these derivation trees will be an attribute value matrix (avm), unlike the baseline, these avms capture the information stored in the gloss. A simplified example avm is presented in Figure 6.2.2.

These initial grammars are further generalized by using rule compression, creating better performing grammars. Rule compression was run at six different levels: No compression, 100%, 80%, 60%, 40%, and 20% input overlap. The rest of data presented here will use the grammars from the boosted tests. As expected this did result in an increase of coverage for both languages, as presented in Table 6.8 for Tagalog and Table 6.9 for Turkish. Even with minimal rule compression the system outperformed the naive baseline for both languages. Although the few forms in Tagalog which the baseline identified which the system could not parse, still were unparsed after compression, because of lacking the proper verbal root.

There is a plateau which occurs with the higher percentages of required input overlap, this is most noticeable in the Tagalog data. This is the result of having many rules which only have a single instance in the data, so they only have one input. A position class with only one input can only have an input overlap of 1/N, where N is a positive integer. So in these cases the many position classes which were not compressed at the 100% level were not able to be compressed until getting below 50%.

One more benefit of combining position classes is the compactness of information as presented to the Grammar Matrix questionnaire. This is generally measured in the number of choices used to define a choices file, where a choice is any line which specifies a piece of information. Typically for both languages the greatest gain in reducing the number of choices needed was also for the highest input overlap percentage.

% input overlap	No. Choices	Verb Types	Position Classes	Coverage
Baseline	N/A	N/A	N/A	0.803
No Compression	1445	**43**	62	0.794
100%	1356	43	22	**0.824**
80%	1356	43	22	0.824
60%	1356	43	22	0.824
40%	1346	43	17	0.824
20%	**1342**	43	**15**	0.824

Table 6.8: Coverage of Test Data for Tagalog

% input overlap	No. Choices	Verb Types	Position Classes	Coverage
Baseline	N/A	N/A	N/A	0.459
No Compression	3975	**137**	168	0.630
100%	3716	137	52	0.668
80%	3696	137	42	0.668
60%	3696	137	42	0.668
40%	**3639**	137	**18**	**0.674**
20%	3639	137	18	0.674

Table 6.9: Coverage of Test Data for Turkish

Like the French data the most significant gain is seen when combining the most similar position classes (those with 100% input overlap). While lower levels of required overlap were tested for the sake of completeness, it is likely these will generate rules which don't represent the underlying forms as accurately. Although in many cases the grammar fails to parse test cases based on the lack of lexical entries not the morphological conjugation.

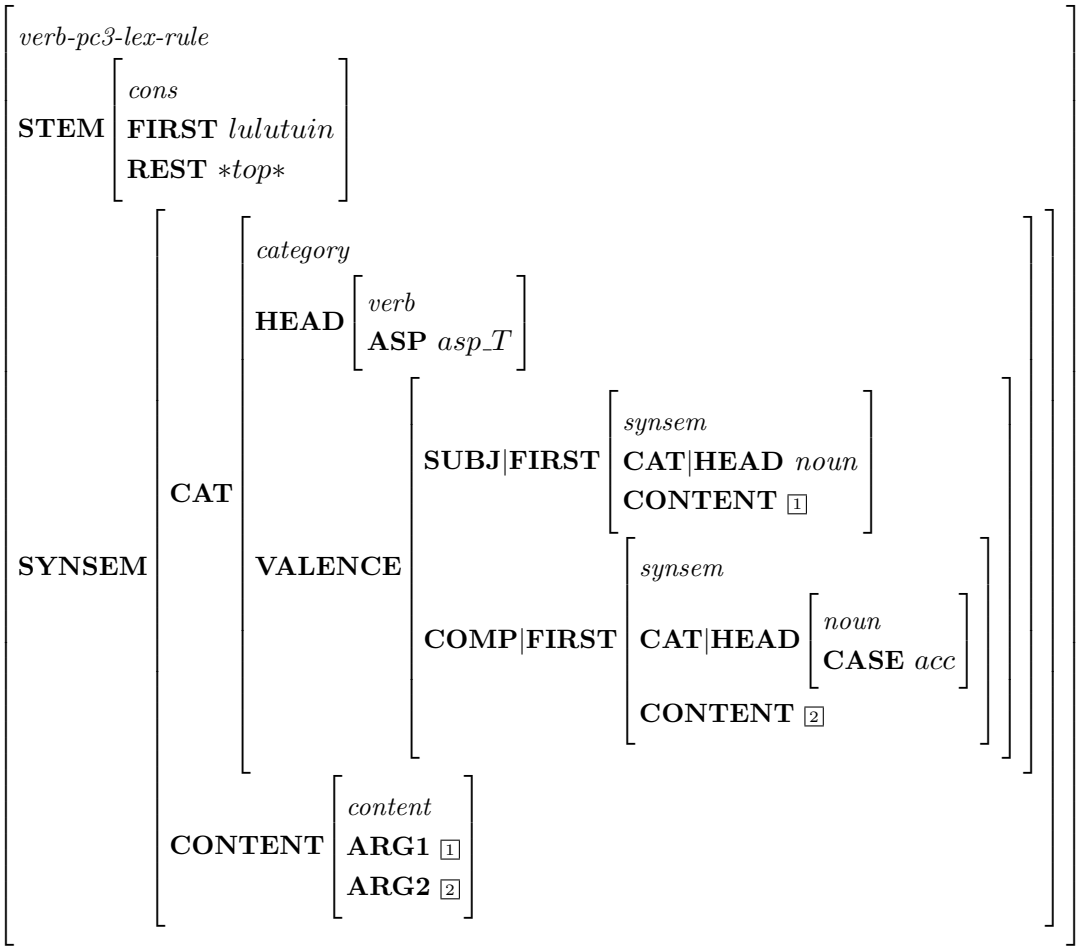

Figure 6.4: Example of Simplified AVM for *Lu-lutu-in* as Parsed by MOM-generated Grammar

6.3 Summary

This section presents the results for the different parameters explained in Chapter 4. The means of evaluation for each step was explained, and the results for each step are analyzed and presented to demonstrate the optimum configurations for generating meaningful grammars. The French data showed that even with a small amount of IGT many forms can be generalized from a small sample, which resulted in grammars able to parse 80% of the forms when only shown 1% of the data. The ODIN data showed that even with noisier data, the system can generate grammars which would parse 67-82% of held out test cases.

Chapter 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

The system presented here provides a means for automatically generating HPSG fragments by utilizing two linguistically rich resources: the Grammar Matrix customization system (Bender & Flickinger, 2005; Drellishak, 2009; Bender et al., 2010b) and the Online Database of INterlinear text (Lewis, 2006). The IGT works as the input for the system providing it with the data needed, and a choices file is output for automatic grammar creation and testing.

The potential of the system is demonstrated using a set of very clean and comprehensive set of data in French. These tests showed that even with a small amount of data the process could generate high coverage grammars. By working with ODIN data, the system demonstrated a potential for working with real world data and varying languages. Even with noisy data, grammars could be generated which would parse 67-82% of verbs from test data.

7.2 Future Work

This study opens the way for many directions and improvements. Identifying noise in the system and either correcting or omitting these cases would be an important part of moving forward with this project.

Better understanding and processing of the grams in the gloss line would help to improve the system as well as the grammars output. One method would be classifying each gram of the gloss line as being a predicate or feature before performing alignments. While it would have a smaller effect on the coverage, it would greatly increase the usefulness of the output grammar. One part of this effort could be leveraging more linguistic cues that a set

of tokens are related across the lines of IGT. Another method for better dealing with grams is to extend the system to identify the features and refer the information to the relevant to sections of the Grammar Matrix questionnaire, such as Number, Person, Gender, Tense, etc.

Missing morpheme segmentation in either the gloss or language line can create problematic alignments for the system. The addition of code to detect and better divide missing morpheme segmentation would help to maintain clear alignments, perhaps by matching tokens heuristically with those which are already well segmented.

Another aspect would be using the same architecture with other parts of speech that can be handled by the Grammar Matrix customization system, such as nominal morphology. The system was designed to be partially part of speech independent, and with minor additions could handle these new cases.

REFERENCES

Bender, E. M. (2007). Combining Research and Pedagogy in the Development of a Crosslinguistic Grammar Resource. In T. H. King & E. M. Bender (Eds.), *Proceedings of the geaf07 workshop* (pp. 26–45). Stanford, CA: CSLI. Retrieved from http://csli-publications.stanford.edu/GEAF/2007/geaf07-toc.html

Bender, E. M. (2008). Grammar Engineering for Linguistic Hypothesis Testing. In A. P. Nicholas Gaylord & E. Ponvert (Eds.), *Proceedings of the texas linguistics society x conference: Computational linguistics for less-studied languages* (pp. 16–36). Stanford: CSLI Publications ONLINE.

Bender, E. M., Drellishak, S., Fokkens, A., Goodman, M. W., Mills, D. P., Poulson, L., & Saleem, S. (2010b). Grammar Prototyping and Testing with the LinGO Grammar Matrix Customization System. In *Proceedings of the acl 2010 system demonstrations* (pp. 1–6). Uppsala, Sweden: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/P10-4001

Bender, E. M., Drellishak, S., Fokkens, A., Poulson, L., & Saleem, S. (2010a). Grammar Customization. *Research on Language & Computation*, *8*(1), 23–72. Retrieved from http://dx.doi.org/10.1007/s11168-010-9070-1 (10.1007/s11168-010-9070-1)

Bender, E. M., & Flickinger, D. (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, E. M., Flickinger, D., & Oepen, S. (2002). The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In J. Carroll, N. Oostdijk, & R. Sutcliffe (Eds.), *Proceedings of the workshop on grammar engineering and evaluation at the 19th in-*

ternational conference on computational linguistics (pp. 8–14). Taipei, Taiwan.

Bickel, B., Comrie, B., & Haspelmath, M. (2008). *The Leipzig glossing rules: Conventions for interlinear morpheme-by-morpheme glosses.* Retrieved from `http://www.eva.mpg.de/lingua/resources/glossing-rules.php` (Max Planck Institute for Evolutionary Anthropology and Department of Linguistics, University of Leipzig)

Bozsahin, C. (2002). The Combinatory Morphemic Lexicon. *Computational Linguistics*, *28*(2), 145–176.

Charniak, E. (1997). Statistical Parsing with a Context-free Grammar and Word Statistics. In *Proceedings of the fourteenth national conference on artificial intelligence.*

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars.* Stanford: CSLI.

Creutz, M., & Lagus, K. (2006). Morfessor in the Morpho Challenge. In *Pascal challenge workshop on unsupervised segmentation of words into morphemes.* Venice, Italy.

Drellishak, S. (2009). *Widespread but Not Universal: Improving the Typological Coverage of the Grammar Matrix.* University of Washtington.

Farrar, S., & Lewis, W. D. (2007). The GOLD Community of Practice: an infrastructure for linguistic data on the Web. *Language Resources and Evaluation*, *41*(1), 45-60. Retrieved from `http://dx.doi.org/10.1007/s10579-007-9016-x` doi: 10.1007/s10579-007-9016-x

Flickinger, D. P. (2000). On Building a More Efficient Grammar by Exploiting Types. *Natural Language Engineering*, *6*(1), 15–28.

Lewis, W. D. (2006). ODIN: A Model for Adapting and Enriching Legacy Infrastructure. In *Proceedings of the e-humanities workshop, held in cooperation with e-science 2006: 2nd ieee international conference on e-science and grid computing.* Amsterdam. Retrieved from `http://faculty.washington.edu/wlewis2/papers/ODIN-eH06.pdf`

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *COMPUTATIONAL LINGUISTICS*, *19*(2), 313–330.

Miller, P. H., & Sag, I. A. (1997). French Clitic Movment without Clitics or Movement. *Natural Language and Linguistic Theory*, *15*, 573–639.

Monson, C., Llitjs, A. F., Ambati, V., Levin, L., Lavie, A., Alvarez, A., ... Probst, K. (2008). Linguistic Structure and Bilingual Informants Help Induce Machine Translation of Lesser-Resourced Languages. In *Proceedings of the sixth international conference on language resources and evaluation.*

Och, F. J., & Ney, H. (2003). A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, *29*(1), 19–51.

Oepen, S. (2001). [incr tsdb()] — *Competence and Performance Laboratory. User Manual* (Technical Report). Saarbrücken, Germany: Computational Linguistics, Saarland University. (in preparation)

Oflazer, K., Nirenburg, S., & Mcshane, M. (2001, March). Bootstrapping Morphological Analyzers by Combining Human Elicitation and Machine Learning. In *Computational linguistics* (Vol. 27, pp. 59–86). Retrieved from http://www.mitpressjournals.org/doi/pdf/10.1162/089120101300346804

Pollard, C. J., & Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.

Rackowski, A., & Richards, N. (2005). Phase Edge and Extraction: A Tagalog Case Study. *Linguistic Inquiry*, *36*(4), 565–599.

Schapire, R. E. (2003). The Boosting Approach to Machine Learning: An Overview. In *Mathematical sciences research institute workshop on nonlinear estimation and classification.*

Siegel, M., & Bender, E. M. (2002). Efficient Deep Processing of Japanese. In *Proceedings of the 3rd workshop on asian language resources and international standardization at the 19th international conference on computational linguistics.* Taipei, Taiwan.

Xia, F., & Lewis, W. D. (2007). Multilingual Structural Projection across Interlinearized Text. In *Proceedings of NAACL HLT 2007* (pp. 452–459). Rochester, NY. Retrieved from http://aclweb.org/anthology//N/N07/N07-1057.pdf