Generation of Machine-Readable Morphological Rules from Human-Readable Input *

Michael Wayne Goodman

Department of Linguistics, University of Washington,
Box 354340 Seattle, WA 98195-4340, USA,
goodmami@uw.edu

Summary. This paper presents a new morphological framework for a grammar customization system. In this new system, the Lexical Integrity Principle is upheld with regards to morphological and syntactic processes, but the requirements for wordhood are expanded from a binary distinction to a set of constraints. These constraints are modeled with co-occurrence restrictions via flags set by lexical rules. Together with other new features, such as the ability to define disjunctive requirements and lexical rule hierarchies, these co-occurrence restrictions allow more complex, long-distance restrictions for phenomena such as bipartite stems, inclusive and exclusive OR-patterned restrictions on morpheme occurrence, constraints dependent on lexical type, and more. I show how the system is able to correctly handle patterns of French object clitics, Lushootseed tense and aspect markers, and Chintang bipartite stems.

Keywords: morphology, morphotactics, grammar engineering, HPSG

Contents

1 Introduction **2**
- 1.1 Inflectional Morphology . . . 3
- 1.2 Computational Grammars . . . 3
- 1.3 Grammar Creation and Customization . . . 4
- 1.4 Overview . . . 4

2 Background **4**
- 2.1 Head-Driven Phrase Structure Grammar . . . 4
- 2.2 The LinGO Grammar Matrix . . . 5
- 2.3 Paradigm Function Morphology . . . 5

* Special thanks to Emily Bender, Olivier Bonami, Francis Bond, Gregory Stump, Fei Xia, an anonymous reviewer, and others for their comments, suggestions, and insightful questions as the work progressed.

University of Washington Working Papers in Linguistics (UWWPL). Volume 30 (2013).

3 The Grammar Matrix and Morphology **6**
3.1 The Questionnaire . . . 7
3.2 Terminology . . . 7
3.3 The Matrix Core and Lexical Rules . . . 8
3.4 Form and Meaning . . . 9
3.5 Lexical Rule Hierarchy . . . 11

4 The Morphotactic Framework **12**
4.1 INFLECTED and Lexical Integrity . . . 12
4.2 Position Classes and Rule Inputs . . . 16
4.2.1 Inputs . . . 16
4.2.2 Implicit Inputs . . . 17
4.2.3 Input Supertypes . . . 17
4.3 Co-occurrence Restrictions . . . 18
4.3.1 Flags and Values . . . 18
4.3.2 Require and Forbid Restrictions . . . 19
4.3.3 Flags, Lexical Rules, and Disjunctions . . . 21
4.3.4 Obligatoriness . . . 21

5 Evaluation and Test Cases **22**
5.1 Parsing Performance . . . 22
5.2 Lushootseed . . . 23
5.3 French Pronominal Affixes (or Clitics) . . . 24
5.4 Chintang Bipartite Stems and Free Prefix Ordering . . . 25
5.5 Ease of Human Effort . . . 26

6 Related Work **27**
6.1 Early Methods of Computational Morphology . . . 28
6.2 Flag Diacritics . . . 29
6.3 Feature-Driven Morphotactics . . . 30

7 Conclusion **31**

1 Introduction

This paper describes a new morphology component for the LinGO Grammar Matrix customization system (Bender et al., 2010b), which is a tool for the semi-automatic creation of implemented HPSG grammars. The task is defined as follows: given user-provided information about the morphology of a language, such as a listing of lexical rules and their relationships to each other and to lexical types, create a machine-readable morphological model of the language that faithfully encodes those relationships. This task entails taking high-level description and crafting valid grammar types, rules, features, and values, as well as placing them into appropriate hierarchies (reusing structures already

established in the grammar when possible), and then integrating these structures into the grammar produced by the customization system. This work also includes the design of an easy-to-use user interface for describing morphological models.

Following O'Hara (2008), I conceptualize this task mostly as problem of **morphotactics**, separating the ordering and co-occurrence restrictions of morphemes from both their syntactic and phonological consequences. This approach puts the lexical rules and their relationships to each other as my prime concern, although the system does also address morphosyntax, morphosemantics, and to a lesser extent morphophonology.

The design and implementation of the system has a number of challenges. First of all, I will attempt to maintain a balance between the intuitiveness of describing a morphological model and the constraints imposed by the mechanical framework in which it is implemented. Aiming to model any human language, the system should allow complex sequences of lexical rules, and co-occurrence restrictions between them, to be defined. Because the resulting grammar may later be hand-developed, unnecessary complexity should be minimized and the resulting machine-readable grammar should be human-understandable.

1.1 Inflectional Morphology

Morphology is traditionally separated into two subsystems: inflection and derivation. **Inflectional morphology** is concerned with associations among lexemes, morphemes, and syntactic or semantic properties. For example, *walk* and *walked* are of the same lexeme, WALK, but the latter has a suffix *-ed* and a semantic property for past-tense, while the former does not. The base meaning has not changed—the word is still the verb *to walk*, but with an additional semantic property. **Derivational morphology**, however, results in a different base meaning or lexical category. For instance, the *-er* suffix on a verb results in a noun that represents the agent of the original verb, such as *walker* from *walk*. Compounding is a derivational process[1] where two independent words (or **unbound morphemes**) join together to form a new word, such as *spacewalk* from *space* and *walk*. The distinction between inflectional and derivational morphology is not always clear, and it has even been suggested that they exist on a lexical-derivational-inflectional continuum (Bybee, 1985), where the contrast is affected by things like obligatoriness of expression and the strength of the semantic change. While I don't intend to take a strong opinion on this theoretical issue, the processes covered by this work are arguably only inflectional.

1.2 Computational Grammars

In this work, a **grammar** is a model of the behavior[2] of a language. Descriptive linguists construct grammars to capture the behavior of interesting linguistic phenomena. Computational linguists create or implement grammars in a programming language or some other formalism, and the grammars can be used to parse and generate valid sentences in the language. Some computational grammars are extracted from language corpora using statistical methods, and others are compilations of hand-written rules. This work is concerned with hand-written, rule-based computational

[1] Although some may place it somewhere closer to syntax.

[2] More specifically, the structures that allow grammatical sentences and disallow ungrammatical sentences.

grammars, although even within this realm there have been fruitful efforts to derive the rules automatically (Bender et al., 2012b).

1.3 Grammar Creation and Customization

As the process of writing a grammar by hand is long and difficult, various tools have been written to aid in grammar development. **Grammar customization** (Bender et al., 2010b) is a process where a core grammar (common to a set of—perhaps all—languages) is augmented with rules and structures specific to the language being described. My work builds on such a grammar customization system.

1.4 Overview

In this paper, I will describe how a user-provided description of a morphological system is interpreted by my framework and how it is transformed into the resulting HPSG implementation. I review some background work in Section 2 and place my work in those contexts. In Section 3, I describe how the Grammar Matrix deals with morphology in general, from the user-centric questionnaire to the digitized definitions of lexical rules. Section 4 is the primary section describing the workings of my morphotactic framework, such as the interaction of lexical rules with phrase-structure rules, position classes and lexical rule hierarchies, and co-occurrence restrictions. The performance is evaluated and the new functionality of my framework is tested with exemplar language phenomena in Section 5. In Section 6 I look at some related work and compare their approaches to my own.

2 Background

This section serves to place my work in its theoretical and implementational contexts; namely, the theory and framework of Head-driven Phrase Structure Grammar (HPSG; in Section 2.1) and the project and codebase of the LinGO Grammar Matrix (in Section 2.2). Finally, in Section 2.3 I bring up Paradigm Function Morphology in order to situate my work to current theoretical morphology.

2.1 Head-Driven Phrase Structure Grammar

The Grammar Matrix produces grammars which are couched in the framework of Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag, 1994) and map strings to semantic representations in the format of Minimal Recursion Semantics (MRS, Copestake, 2002a). These grammars are written in the TDL formalism, as interpreted by the suite of software produced by the DELPH-IN[3] collaboration.

For present purposes, the most salient features of HPSG are as follows: It is a surface-oriented, constraint-based framework, giving grammars which are suitable for both parsing and generation. HPSG grammars consist of **types** and **instances**. **Instances** include **lexical entries**, **lexical rules** and **phrase structure rules**. Each of these are considered Saussurean signs, which pair constraints on form with constraints on meaning. The **types**[4] capture the constraints which are shared across

[3] `http://www.delph-in.net`

[4] These types include lexical types, lexical rule types, and phrase structure rule types, as well as ancillary types used in the definition of the others.

different entries. The types are organized into a **type hierarchy** such that constraints on supertypes are shared by their subtypes. My main focus in this paper will be on the types and instances for lexical rules, and their interaction with the types and instances for lexical entries.

In a DELPH-IN-style HPSG derivation, the lexical rules are non-branching productions at the bottom of a phrase structure tree. Each node in the tree is licensed by a lexical item, lexical rule, or phrase structure rule, and the nodes are labeled with feature structures whose features and values reflect the constraints on the rules used to license the structure.

2.2 The LinGO Grammar Matrix

The LinGO Grammar Matrix customization system (Bender et al., 2002, 2010b) is a web-based software system for creating implemented HPSG (Pollard and Sag, 1994) grammar fragments on the basis of user input of typological and lexical information. These fragments are compatible with the LKB grammar development environment (Copestake, 2002b) and other DELPH-IN software. The system consists of a core grammar (types and constraints hypothesized to be useful across all languages) and a series of "libraries" of analyses of recurring, but non-universal, phenomena. Most existing and future Matrix libraries involve morphological expression in at least some languages, so the customization system must provide a means for users to define lexical rules which attach the morphemes expressing various linguistic features.

Grammar customization is a process that first involves eliciting typological information about a language from a user. The Grammar Matrix web-based questionnaire provides some explanations of linguistic phenomena and questions designed to guide the user through the description of an analysis. When the questionnaire has been sufficiently filled out, the user may choose to "customize" a grammar or test by generation. Choosing to customize a grammar causes the information provided to be used in the automatic construction of an implemented HPSG grammar, which is then presented to the user as a downloadable file (Bender et al., 2002, 2010b). Choosing to test by generation will perform the customization in the background, then use the resulting grammar with a generator and a set of semantic templates to realize strings deemed grammatical by the grammar. The user can then use this output to further refine their choices regarding the language.

2.3 Paradigm Function Morphology

One of the recent advances in theoretical morphology is Stump (2001)'s Paradigm Function Morphology (PFM). Stump first describes a typology of morphological theories along two dimensions: lexical versus inferential, and incremental versus realizational. **Lexical** morphologies list morphemes with their morphosyntactic properties in the lexicon, whereas **inferential** morphologies define relations between roots and inflected forms with lexical rules. **Incremental** morphologies accumulate morphosyntactic properties as morphemes are added, whereas in **realizational** morphologies inflected forms are licensed by the morphosyntactic properties a word is associated with. Stump argues for a inferential-realizational model, such as his theory of Paradigm Function Morphology.

The approach described in this paper is inferential-incremental. Some of Stump's arguments for realizational morphologies instead of incremental are that they more easily allow multiple exponence (multiple changes in wordform yield the same change in semantics) and zero realization (no

change in wordform yields a change in semantics), but neither of these are problematic for my system. Multiple exponence works because constraints are unified—if multiple rules apply the same morphosyntactic (or semantic) properties, they are simply "merged" together. Zero realization is not a problem because zero-marked rules are allowed, thus a developer may write a rule that adds semantic property, such as singular number, without any accompanying change in wordform.[5] That is, while there is a rule for every syntactic or semantic contribution to a word, and a rule for every orthographic/phonological contribution, neither syntactic/semantic nor orthographic/phonological contributions are required elements of the lexical rules.

The paradigm functions of PFM essentially map a word root and its full set of properties to the fully-inflected wordform (the appropriate cell in that word's paradigm corresponding to the property set). Paradigm functions themselves are defined by realizational rules, which map a subset of properties with the corresponding realization in the wordform. Realization rules are grouped in rule blocks, such that only one rule per block can apply on a single word. Rule blocks are labeled, but unordered, and the paradigm functions define an order. There is also an implicit, universal Identity Function Default that maps any set of properties to the same wordform. It applies if no other rule in a block has been applied. Rules in a block are selected by Panini's principle—the rule with the narrowest interpretation is applied.

Paradigm functions don't have a direct analog in my system, but their functionality is covered in the way that position classes specify lexical types as inputs, and they themselves define groups of mutually exclusive rules (similar to rule blocks). Paradigm functions differ in that they also define the ordering the lexical rules for each lexical type. In my system, rule ordering is defined by the chain of inputs specified on the position classes. PFM does not have explicit support for co-occurrence restrictions, but the mechanisms in place for rule ordering, grouping, and selection seem to be sufficient. The Identity Function Default is enviable, since my system still needs to posit empty rules to account for lexical rules that don't change the form. PFM seems to be a great solution for generation, in that the full set of properties is available at each rule application, but it is not obvious how well it would work in analysis.

3 The Grammar Matrix and Morphology

The morphotactic framework described in this work builds on existing morphological infrastructure in the Grammar Matrix. The user-facing web-based questionnaire is expanded to accommodate the changes, and is described in Section 3.1. Definitions of terminology used in later sections are given in Section 3.2. The relevant grammar types and rules given in the Grammar Matrix core grammar are described in Section 3.3. Section 3.4 explains how lexical rules in Matrix-based grammars affect wordform and meaning (syntax and semantics). Finally in Section 3.5 I describe how lexical rule hierarchies are created and used with the system.

[5] Note that other grammatical processes, such as agreement, would limit the properties that could be applied to a word, so this does not imply that any and all zero-marked rules necessarily apply

3.1 The Questionnaire

The Grammar Matrix questionnaire[6] is a web form comprised of many pages—roughly one page for each linguistic phenomenon covered (such as Number, Case, Tense and Aspect, Argument Optionality, etc.). Questions on each page can be single typological choices, such as how a grammar marks number or case, or unbounded lists of entries for things like lexical items and rules.

Figure 1 shows a screenshot of a portion of the Morphology subpage (including my additions), which contains information for marking singular and plural number on English common nouns. The position class has fields relevant to its position in the formation of a word (inputs, whether it is prefixing or suffixing) and relevant to all the lexical rules it contains (such as co-occurrence restrictions). The lexical rules it contains may be defined as a hierarchy (using by specifying the "supertypes" field), in cases where rules can be grouped by common features or co-occurrence patterns. Lexical rules types define the syntactic and semantic features they provide, as well as any co-occurrence restrictions applicable only to them and their descendants. Further, they can define zero or more lexical rule instances. A lexical rule instance defines a rule's orthographic contribution (if any). A lexical rule without any instances cannot be applied directly to a word, but its subtypes with instances can. More information on the position class and lexical rule type fields is provided in subsequent sections.

When the user submits the content on a subpage, it is serialized into a choices file. Figure 2 shows the portion of the choices file pertaining to the information in the screenshot of Figure 1, plus, for illustrative purposes, some choices for the common noun lexical type and *dog* and *cat* lexical entries under this type. While the format of a choices file is designed primarily for machine consumption, it is not difficult for a human to decipher its meaning. A secondary purpose for the choices files is that users can save or load them into the questionnaire, so they can work on multiple sessions or resume a previously started session.

The questionnaire may change from time to time, but the main point is that it is a user-focused form that asks questions about a grammar mostly independent of the formalism of the resulting implemented grammar. In other words, it represents an abstract description of the grammar, and the user need not know HPSG or TDL in order to produce a working grammar.

Finally, it is worth noting the test-by-generation feature (Bender et al., 2010a) of the questionnaire. While it is not directly relevant to the computation of morphological models, it is exceptionally useful in testing the effects of a user's choices. While the user could customize the grammar, download it, load it into a parser, and parse some test sentences, viewing generation results within the questionnaire saves a large amount of time. Moreover, testing using generation instead of parsing helps the user more easily find problems in the grammar, since it generates all possible outputs allowed for a semantic input to the generator. For any significantly complex morphological system, testing by generation should be an important part of the workflow.

3.2 Terminology

Now that the basic structures and entities of the system have been introduced, this is a good point at which to clarify terminology used in this paper. Table 1 lists some terms that may take a

[6] Available at `http://www.delph-in.net/matrix/customize`

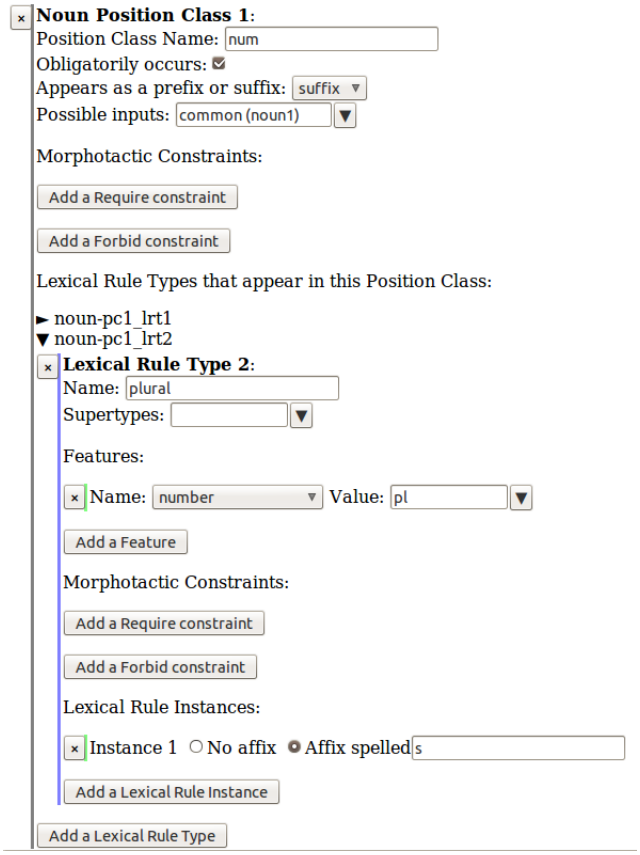

Figure 1: Screenshot of editing lexical rules in the questionnaire

particular meaning for this paper. While some of these descriptions may be vague now, the terms will be further used and refined in the later sections.

3.3 The Matrix Core and Lexical Rules

Customized grammars use information provided by a user about a grammar, but they build on the generic types and rules defined in the Grammar Matrix Core. The existing Matrix (on top of which my work builds) provides a series of types for modeling lexical rules, the most general of which is *lex-rule*. This type (in keeping with the Minimal Recursion Semantics approach (Copestake et al., 2001) to semantic compositionality) constrains lexical rules such that they can add but not remove semantic information. In addition, it specifies a daughter feature which can be thought of as the **input** of the rule. On *lex-rule*, the type of the daughter feature is constrained only to be a *word-or-lexrule*, meaning that unless further constraints are applied, a grammar's lex-rules can apply to any stem and in any order. The Grammar Matrix provides subtypes of *lex-rule* along two dimensions.

```
noun1_name=common
noun1_feat1_name=person
noun1_feat1_value=3rd
noun1_det=obl
noun1_stem1_orth=cat
noun1_stem1_pred=_cat_n_rel
noun1_stem2_orth=dog
noun1_stem2_pred=_dog_n_rel
noun-pc1_name=num
noun-pc1_obligatory=on
noun-pc1_order=suffix
noun-pc1_inputs=noun1
noun-pc1_lrt1_name=singular
noun-pc1_lrt1_feat1_name=number
noun-pc1_lrt1_feat1_value=sg
noun-pc1_lrt1_lri1_inflecting=no
noun-pc1_lrt2_name=plural
noun-pc1_lrt2_feat1_name=number
noun-pc1_lrt2_feat1_value=pl
noun-pc1_lrt2_lri1_inflecting=yes
noun-pc1_lrt2_lri1_orth=s
```

Figure 2: Portion of a choices file for English

The first dimension is for rules that inflect with a change to the wordform (*infl-lex-rule*) or for zero-expression rules (*const-lex-rule*). The second dimension is for rules that constrain different kinds of semantic or syntactic properties (e.g. *add-only-no-ccont-rule*, *val-change-only-lex-rule*, *cont-change-only-lex-rule*, etc.). These rule types copy the information that is unchanged, and allow a subset of the features to be affected by the rule. Through multiple inheritance, there is a wealth of possible combinations of rules from these two dimensions.

Regarding the specification of rule inputs, one may appeal to the notion of productivity and say that the pairing of lexemes with lexical rules is not categorical, but rather the acceptability of the pairing lies on some scale. For example, the user may not want to claim that only one plural form of "octopus" is correct (of, say, "octopuses", "octopodes", or "octopi"), but that all are possible and that some are more preferable than others. I offer no mechanism for quantifying acceptability, but neither do I require users to make such exclusive decisions. The user may offer all possibilities as options, although more open-ended models will be less efficient in parsing and generation. I delegate the rating of acceptability to extrinsic processes.

3.4 Form and Meaning

In the linguistic literature, a **morpheme** was traditionally defined as the smallest unit pairing of form and meaning.[7] That is, some change in the wordform occurs in tandem with the addi-

[7] An excellent description of **morphemes** and other relevant terminology in this section can be found in Haspelmath and Sims (2010).

Term	Description
lexeme	a triple of <ORTHOGRAPHY, LEX-TYPE, PREDICATE>
orthography	the form, or spelling, of a lexeme
lex(ical)-type	a grammar type used to describe lexemes
predicate	the semantic predicate contributed by a lexeme
lex(ical)-item	a lexeme listed in the lexicon
lex(ical)-object	in parsing or generation, an object composed of a lexeme and the effects of any accumulated lexical rules
lex(ical)-rule	a grammar rule that affects the form and/or syntactic or semantic features of a lexical object; also, a generic term for lex-rule types and lex-rule instances
lex(ical)-rule type	a grammar type that places syntactic or semantic features on lex-rules
lex(ical)-rule instance	a lex-rule that inherits from a lex-rule type and may apply to a lexical object and affect wordform
morphotactic unit	a lex-type or lex-rule; that is, something that may place or satisfy co-occurrence restrictions
root	a lex-object that has no lex-rules applied to it
stem	a lex-object that may serve as input to a lex-rule
word	a lex-object with all necessary inflection for use in a phrase-structure rule
position class	a set of lex-rules mutually exclusive in word-formation that share a "slot", or a location of possible application
input	stems that may serve as input to lex-rules; used to specify the position of a position class

Table 1: Terminology used in this paper

tion of syntactic or semantic properties to a stem. Modern theoretical morphology has shifted away from this terminology because changes to words may occur without changes to the form (**zero-expression**), without adding semantic properties (**empty morphemes**), or with the addition of multiple properties (**portmanteaus**).[8] Moreover, there are often alternations in the form of a morpheme—**allomorphs**—while the semantic property remains the same, and it is inelegant to define them as distinct morphemes. For example, there are many ways to mark the English plural (*-s*, *-es*, *-en*, *-ren*, etc.), but they are better analyzed as variations of the same morpheme.

The Grammar Matrix, and more generally LKB-style grammars, encode the modification of form with two basic rule types: `%prefix` and `%suffix`. These rules can match the starting or ending characters, respectively, and replace them with others, or they can match nothing and simply append characters. For example, the English Resource Grammar (Flickinger, 2000) has the suffixing rule[9]

[8] However, even these constructs are not without controversy, as some may contend they only exist to support concatenation-only models of morphology (Haspelmath and Sims, 2010), and, indeed, in realizational models (see Section 2.3) they would not be necessary. The Grammar Matrix assumes an incremental morphological system, so for our purposes these constructs are still relevant.

[9] For brevity I do not show the rule's identifier, type, or other syntactic constraints.

for pluralization shown in Figure 3.

```
%suffix (!s !ss) (!ss !ssses) (es eses) (ss sses) (!ty !ties)
        (ch ches) (sh shes) (x xes) (z zes)
```

Figure 3: Suffixing rewrite rule for plural number in the ERG.

There are many variations here based on the preceding characters. Exclamation marks (!) are for character classes, so for instance `!s` is defined (elsewhere) to be all characters except "s". Thus, words ending in something other than "s" will have an "s" appended while preserving the original character. Characters in the `!t` class followed by a "y" will result in the same `!t` character followed by "ies". As my work is focused on morphotactics and not morphophonology, I do not allow the user to write complicated `%prefix` or `%suffix` rules in the questionnaire,[10] but rather allow them to define a simple prefix or suffix that will be appended without any rewriting or character matching. The applicability of rules is defined by matching lexical types or the occurrence of other lexical rules. As it is, the `%prefix` and `%suffix` rules are very basic, and while they may be sufficient for simple morphologies, I suggest that developers use external morphological processors to rewrite word forms (c.f. Bender and Good, 2005). For example, rules defined by my system would add gloss-style additions to the wordform (e.g. "dog-PL"), and the external processor would rewrite this to the surface form (e.g. "dogs").

3.5 Lexical Rule Hierarchy

Grammar types in the Grammar Matrix exist in a hierarchy, and this includes lexical rule types. The questionnaire allows the user to define custom hierarchies of lexical rules within position classes.[11] As with other type hierarchies, lexical rule types lower in the hierarchy inherit constraints placed on ancestor types. Not all lexical rule types can be directly applied on a stem—some types merely serve to add constraints that will be inherited by child types. Those that can be applied on stems will have one or more associated lexical rule instances. These instances will specify the orthographic change (if any) that the rule contributes.

Figure 4 shows how the inheritance of constraints can be useful. The first subtypes in the position class *Gen-Num-PC* add a single property, such as setting the GENDER feature to *fem*. The second level of subtypes are the cross classification of the two binary-valued features, but notice that no new properties are specified; they already inherited the two from their parents. If all the user cared about was the lower, cross-classified types, then the first level of subtypes would be unnecessary (and two properties would be specified on each leaf type). It may be, however, that a valid wordform can, in some cases, only be found when a single feature (masculine, feminine, singular, or plural) is specified, or that further inflection can only happen with plural items. If this were the case, then these morphotactic constraints could be defined on the more general subtypes, rather than the leaf types.

[10] Although nothing prevents them from writing such rules themselves in subsequent development.

[11] Position classes will be further defined in Section 4.2, but for now just assume they are sets of rules from which a maximum of one may be used in a word.

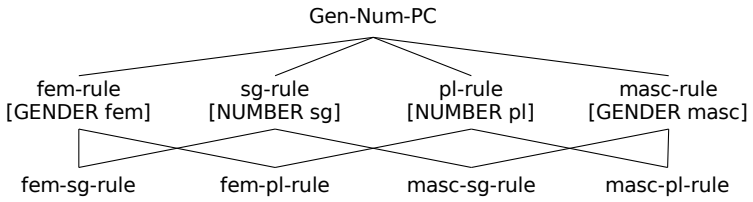

Figure 4: An example lexical rule hierarchy

4 The Morphotactic Framework

In this section I will explain the mechanisms that allow my morphotactic framework to work. Section 4.1 discusses how the Lexical Integrity Principle is upheld, and how I distinguish words that are fully formed versus those that are not. Section 4.2 describes what position classes are and how they are arranged using rule inputs. Section 4.3 covers co-occurrence restrictions and how flags are used with just a handful of possible values to model complex relationships between lexical rules.

4.1 INFLECTED and Lexical Integrity

The primary constraint placed on all phrase-structure rules is based on the **Lexical Integrity Principle** (Bresnan and Mchombo, 1995), which limits exchanges at the morphology-syntax interface: syntax may not inspect word-internal structures, nor can lexical rules directly affect syntactic processes. The lexical object handed off to phrasal rules must be taken as a complete, atomic unit, and thus it must be fully satisfied inflectionally (all required lexical rules have applied). It is important to note that while the distinction between a fully inflected word and an incompletely inflected lexical object is inherently binary (it either is or it isn't), this does not mean that a single, binary flag is all that is needed to model inflectional satisfaction.

The binary word/not-word distinction is handled via types. Lexical rules only allow lexical objects as input, but once a lexical object has been admitted as a daughter to a phrasal rule, it cannot (or rather, its phrasal mother cannot) be used in any other lexical rules. The lexical object that is handed off to phrasal syntax will have a completely inflected wordform and a complete set of syntactic or semantic features, but the phrase-structure rules will be oblivious as to how the wordform was created or how the features were accumulated. Appeals to the lexical integrity principle are not uncommon in HPSG literature (e.g. Manning et al., 1999; Crysmann, 1999), but the discussion tends to focus on morphology-syntax interactions, and there is relatively little discussion around the requirements for wordhood.

The notion of inflectional satisfaction is more complicated. In my system, a grammar may define a required set of morphemes for some lexical type, but there may also be optional morphemes that must or must not co-occur with other morphemes. In other words, inflectional satisfaction is defined as an admissible configuration of applied lexical rules. Since my system is incremental in nature,

the elements for an admissible configuration are accumulated through the application of each lexical rule.

O'Hara (2008) used a binary feature, INFLECTED, to say if a lexical object was inflectionally satisfied (+) or not (-), and a separate matrix of features to model co-occurrence restrictions between morphemes. The phrase-structure rules would only check the value of INFLECTED to determine admissibility, so the separate matrix of features was effectively toothless. More specifically, the lexical rules considered these features, so they could be used to block incompatible lexical rules from applying, but they had no ability to stop an incomplete lexical object from being used in phrase-structure rules. Figure 5 showcases a morphotactic system that is problematic for this system. In the figure, the shaded *LexType* represents a root (uninflected lexeme). Edges represent input paths of rule application. Dashed edges and structures are implicit (generated by the system and not specified by the user; see Section 4.2.2). O'Hara's system allowed lexical rules to specify if they were optional or obligatory with respect to the root, shown here in the boxes representing lex-rules. Rules could require or forbid the occurrence of other rules or lex-types, as *LexRule1* shows.

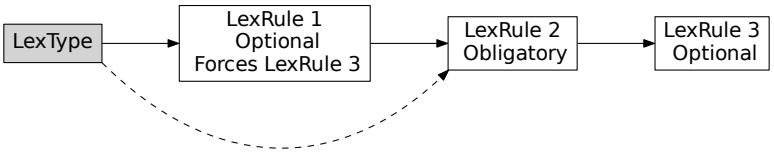

Figure 5: Overlapping Co-occurrence Restrictions

This arrangement is problematic because if the first, optional rule is not applied, INFLECTED should be set to + when the second, obligatory rule applies, but if the first rule does apply, the third rule should set INFLECTED to +, and not the second rule. Figure 6 shows how this otherwise simple arrangement would be inefficiently modeled with boolean values of INFLECTED. Note how two rules must be duplicated.

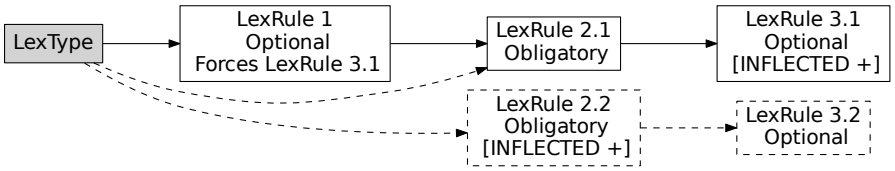

Figure 6: Inefficiently modeling a problematic configuration with boolean values of INFLECTED

From examples like the above, I found that it was impossible to determine a priori if the application of a lex-rule will yield a fully inflected word. Rather, it can only be determined by looking at the complete configuration of applied lexical rules. Therefore, in my system, lexical rules do not set a binary feature to state whether the lexical object is complete, but rather only modify the matrix of

features, and each rule only states its own contribution to, or requirements for, the construction of a word. Phrase-structure rules then deem a lexical object admissible or not by unifying its matrix of features with the satisfactory state (described later). This change is the primary innovation in my work, and the other parts of my system work to support this mechanism.

In implementation, the value of INFLECTED is an AVM (attribute-value matrix) that is defined by the language-specific grammar definitions. This value is a separate grammar type called *inflected*, with a subtype called *infl-satisfied*. All lex-types start with the feature INFLECTED set to the value *inflected*, and can later add constraints on the feature. The type *infl-satisfied* represents the satisfactory state, so any configuration of INFLECTED that unifies with it will succeed when determining the acceptability of words in phrase-structure rules.

Figure 7 shows the unary and binary phrasal rules that phrase-structure rules are based on. Note that the daughters of the rules have INFLECTED specified as *infl-satisfied*. While these constraints are intended to be universal to all implemented grammars, the exact configuration of *infl-satisfied* is specified by each individual grammar. More information about the values of INFLECTED is given in Section 4.3.

$$\begin{bmatrix} \textit{UNARY-PHRASE} \\ \text{ARGS} \quad \left\langle \left[\text{INFLECTED} \quad \textit{infl-satisfied} \right] \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \textit{BINARY-PHRASE} \\ \text{ARGS} \quad \left\langle \left[\text{INFLECTED} \quad \textit{infl-satisfied} \right], \left[\text{INFLECTED} \quad \textit{infl-satisfied} \right] \right\rangle \end{bmatrix}$$

Figure 7: Unary and Binary Phrasal Rules

Example 4.1

Plural and Singular English Nouns:

The following is an example that illustrates the specification of a simple position class and how co-occurrence restrictions are used to require a rule from the position class to apply.

For an English grammar, consider a lexical item for DOG with the lexical type *common-noun*, for which the user wishes to generate the singular form *dog* and the plural form *dogs*. Assuming other parts of the grammar (word order, feature hierarchies, etc.), are already filled out, the user would add information to the questionnaire for the lexical type, its features (e.g. 3rd-person), and its instances (DOG, CAT, etc.). The user would then create an obligatory position class for the number-marking suffix, specify *common-noun* as its input, and add lex-rule types for the plural- and singular-marking rules. The plural lex-rule would specify the number to be plural and have a lex-rule instance that appends "s" to the wordform. Likewise, the singular lex-rule would specify number as singular, but have a zero-marking lex-rule instance. The user-specified choices for this example were given earlier in Figure 2.

The system would read this configuration and posit TDL code for the lexical rules as shown in Figure 8. First is the addenda to the *inflected* and *infl-satisfied* types so they have a flag for marking number. Number-marking is obligatory for common nouns, so the *common-noun-lex* lexical type sets the flag to - (i.e. "unsatisfied"; this mechanism will be further explained in Section 4.3). The supertype of *common-noun-lex* (*obl-spr-noun-lex*) is of no concern here, but note that *common-noun-lex* specifies the person feature (PER) to be *3rd*, and moreover it does not specify number, since number is underspecified until a lexical rule fires. Moving on, notice the position class becomes the grammar type *num-lex-rule-super* and its supertype (*add-only-no-ccont-rule*) allows its rules to add, but not change, constraints to the syntactic features.[12] The position class also satisfies the NUM-FLAG feature, so a common noun marked for number will be available for use in phrase-structure rules. Also generated are *singular-lex-rule* and *plural-lex-rule*, which encode the feature constraints. Not shown are the lex-rule instances, which inherit from these lex-rule types and define the changes to wordform (if any).

```
inflected :+ [ NUM-FLAG luk ].

infl-satisfied :+ [ NUM-FLAG na-or-+ ].

common-noun-lex := obl-spr-noun-lex &
  [ SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG.PER 3rd,
    INFLECTED.NUM-FLAG - ].

num-lex-rule-super := add-only-no-ccont-rule &
  [ INFLECTED [ NUM-FLAG + ],
    DTR common-noun-lex ].

singular-lex-rule := const-lex-rule & num-lex-rule-super &
  [ SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG.NUM sg ].

plural-lex-rule := infl-lex-rule & num-lex-rule-super &
  [ SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG.NUM pl ].
```

Figure 8: TDL output for English number-marking rules

This grammar can generate the singular *dog* and the plural *dogs*. Without the singular rule, the grammar could only generate the plural *dogs*, since the position class is obligatory. If we made the position class optional, the rule could be skipped entirely (thus allowing *dog*), but then the number feature would be ambiguous (underspecified), and the grammar would fail to rule out sentences like **The dog bark*.

[12] The supertype (defined in the Matrix core grammar) enforces this behavior by unifying the syntactic features on the daughter with those of the mother, so any non-subsumptive feature values (i.e. changed values) will cause a unification error.

4.2 Position Classes and Rule Inputs

While morphemes, or lexical rules, can be optional, when they do occur they must occur in a fixed order. This is true across the world's languages with few exceptions (but see Chintang Bickel et al., 2007). As mentioned earlier, the Grammar Matrix uses an incremental morphological system, so lexical rules are applied in sequence and each one contributes some constraints and/or change in wordform. I therefore need an adequate system for specifying rule order. In order to accomplish this, I allow the user to organize lexical rules into type hierarchies rooted in position class types, and they then specify the other types that can serve as input to the position class types.

A **position class** (PC) is a collection of lexical rules, mutually exclusive in word-formation, that appear at a particular position in a word. The rules in a PC are arranged in a type hierarchy, and the type representing the PC is the top-level type of the hierarchy. Thus, lexical rule types in the hierarchy inherit any constraints and input specifications on the PC. To make this clear, Figure 9 shows a position class with the same lex-rule type hierarchy as in Figure 4, but now with inputs specified. A box represents a position class, and edges going into and out of position classes are input paths. Edges within a position class represent inheritance in the lex-rule type hierarchy. Where *gen-num-pc* occurs in the formation of word, only one of the rules in the hierarchy may be applied.

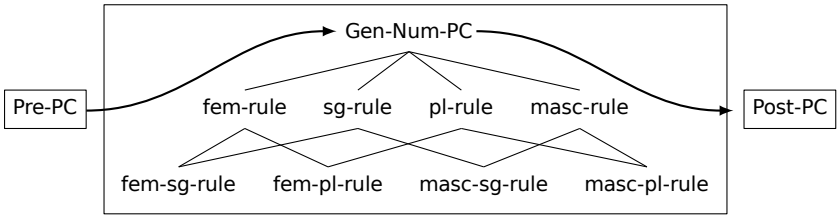

Figure 9: Example lexical rule type hierarchy in a position class

4.2.1 Inputs In my system, an **input** to a PC (and, through inheritance, all its contained lex-rules) is a morphotactic unit that immediately precedes the PC in the sequence of applied rules.[13] It can be any morphotactic unit (including other PCs) not contained by the PC. On the other hand, inputs may only be specified on a PC (i.e. a lexical rule may not specify inputs separate from those of the PC that contains it). Referring back to Figure 9, this means that only *Gen-Num-PC* may specify inputs (in this case, *Pre-PC*), but any type in the hierarchy can be specified as an input to another position class. For example, if only singular lex-rules can precede *Post-PC* (rather than any lex-rule under *Gen-Num-PC*), the input on *Post-PC* could be specified as *sg-rule*). An input is also called a **rule daughter**, since it is specified on the feature DTR.

[13] The order of rule application is not the same as the order of affixes in a word. Each position class may specify whether it is prefixing or suffixing, but otherwise the rules appearing closer to the root have applied earlier than those appearing farther away.

4.2.2 Implicit Inputs The inputs a user specifies are not the only inputs that end up in the resulting grammar. There are also **implicit inputs** that are generated when a position class may optionally occur. Referring back to Figure 5, *Lexical Rule 1* is optional, so there is an arrow (representing an input) that goes around it from *LexType* to *Lexical Rule 2*. This input is not specified by the user, but generated by the system during customization. Note that no implicit input is generated going around *Lexical Rule 2*, because it is obligatory (i.e. required by *LexType*).

4.2.3 Input Supertypes A position class may have more than one input (explicit or implicit), but in the grammar's TDL code, only one input type may be specified. To get around this problem, **input supertypes** are generated, which are the disjunction of all inputs specified for a PC. In TDL, an input supertype is realized as a simple lex-rule type, and it is inherited by all the possible inputs to the PC. The PC then specifies this input supertype as its sole input, and thanks to the inheritance, any of the original inputs may immediately precede the PC.

Example 4.2

Multiple Inputs and Input Supertypes: Spanish Verb Conjugation:

The following is an example that illustrates how a position class defined with multiple inputs is encoded in my system.

Assume a lexical type hierarchy with 3 verb types: *regular-ar-verb*, *regular-er-verb*, and *regular-ir-verb*. The 3rd person singular present tense conjugation is the same for regular *-er* and *-ir* verbs, and this can be modeled with a single position class for the conjugation.[14] The user would create a position class *regular-er-ir-verb-3sg-pres*, and specify its inputs to be *regular-er-verb* and *regular-ir-verb*. This is illustrated in Figure 10:

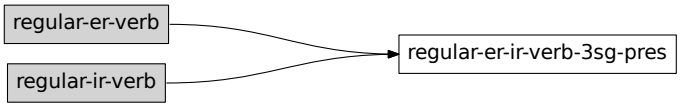

Figure 10: A position class with two inputs

regular-er-ir-verb-3sg-pres is specified by the user to have two possible inputs, but the system needs to specify a single type for its daughter value. When the system customizes the grammar it creates an input supertype, *regular-er-ir-verb-3sg-pres-dtr*, to model this disjunction. Both *regular-er-verb* and *regular-ir-verb* will inherit this intermediate type,[15] as shown in Figure 11, and the position class will now take the intermediate type as its only input, as shown in Figure 12. In this way, both verb types will work with lexical rules defined within the position class.

[14] This may not be the most efficient way to model the distribution, but it suffices for the example.

[15] In addition to other relevant lexical rule supertypes, by virtue of multiple inheritance.

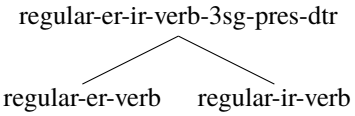

Figure 11: Verb types inheriting from an intermediate type

$$\begin{bmatrix} \textit{REGULAR-ER-IR-VERB-3SG-PRES} \\ \text{DTR} \quad \textit{regular-er-ir-verb-3sg-pres-dtr} \end{bmatrix}$$

Figure 12: A position class specifying an intermediate type as an input

4.3 Co-occurrence Restrictions

In Section 4.1 I introduced the INFLECTED AVM and how it allows a more nuanced description of inflectional satisfaction. The elements that comprise the AVM are attributes called **flags**, and they allow one to require or forbid lexical rules from occurring in particular contexts. Section 4.3.1 describes more explicitly what a flag is and how co-occurrence flags are enforced in general, and section 4.3.2 explains how they are used to require and forbid lexical rules.

4.3.1 Flags and Values Flags are simple attributes with values. They can be assigned values on lexical types and the output of lexical rules, and can be checked on the input to lexical rules. Because I use a single flag to model three states for lexical rules (that it has occurred, that it has not yet and must later occur, and that it is unconstrained (i.e. optional)), I found that boolean values of flags were inadequate. A simple ternary value would capture these three states, but would fail to capture disjunctive pairings, such as the satisfactory state (a flag is satisfactory if the rule has occurred or is unconstrained). Thus, I needed a value hierarchy with three leaves and two disjunctive nodes. Fortunately, this hierarchy already existed in the Grammar Matrix codebase as the *luk*[16] hierarchy, shown in Figure 14. It is a ternary extension of the common boolean values (including *na* in addition to + and -), but I also use the underspecified (disjunctive) types *na-or-+* and *na-or--*. In the context of co-occurrence restrictions, these values can be given more intuitive glosses, as shown in Figure 13.

+	lexical item or rule has occurred
-	lexical rule has not yet occurred, and must occur later
na	lexical item or rule has not occurred, nor has it been constrained
na-or--	initial value
na-or-+	satisfactory value

Figure 13: Intuitive interpretation of *luk* values

Lexical rules specify input and output constraints. Input constraints are used to check for compatibility, and output constraints define the effects the lexical rules have on the lexical objects. All

[16] The *luk* hierarchy is named after the Polish logician Jan Łukasiewicz, and is borrowed from the English Resource Grammar (Flickinger, 2000).

co-occurrence restrictions use a combination of input and output constraints.

All flags that are created have the same initial value and must unify with the same value on *infl-satisfied*. The initial value is *na-or--*, and the satisfaction value is *na-or-+*. With this value for satisfied constraints, either *na* or + will unify, and thus succeed, so the only co-occurrence restrictions that prevent lexical objects from being used in phrase-structure rules are requirement restrictions that leave a flag set to - (described in further detail in the next section). Other co-occurrence restrictions are enforced by the lexical rules themselves (i.e. they won't fire if the relevant flag states are incompatible). Further, since *na-or--* and *na-or-+* share a subtype, they will unify with each other, so flags that have not been constrained will not cause unification errors.

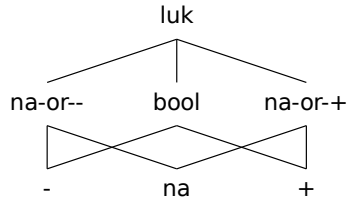

Figure 14: The *luk* value hierarchy

4.3.2 Require and Forbid Restrictions Users are allowed to define two kinds of co-occurrence restrictions in their grammar—REQUIRE and FORBID—but in fact there are three ways these are synthesized in the grammar. The difference is due to the unidirectionality of REQUIRE, while FORBID is bidirectional. It was a design decision to keep REQUIRE as a single co-occurrence restriction from the user's point of view, rather than having a confusing split (moreover, a split that is entirely predictable by looking at rule inputs).

REQUIRE enforces co-occurrence, so if a rule occurs and it requires some other rule, the other rule must also occur. This relationship does not apply in the other direction, so the second rule may still occur even if the first one does not. FORBID disallows co-occurrence, so one rule or the other may occur, but not both.[17] For example, consider two sequential rules, *A* and *B*, where *B* can occur after *A*, that apply on some stem called *stem*. Table 2 explains, and shows the possible outputs, for the three kinds of co-occurrence restrictions. Note that REQUIRE is split into two restrictions, where **require** is used when a later rule enforces co-occurrence with an earlier rule, and **force** is used when an earlier rule enforces co-occcurrence with a later rule.

In the implementation, the flags are represented on the INFLECTED feature, and each rule has two of these AVMS—one in the main body (i.e. the mother) of the rule, and one on the daughter. The daughter functions as the input to the rule, so any constraints placed on DTR.INFLECTED serve as "input constraints", which are just checked for compatible values (i.e. the rule will not fire if the constraints are not compatible with the input's flags). Constraints placed on the rule's primary

[17] But note that in the absence of other constraints, it is also valid if neither rule occurs.

Restriction	Description	Possible outputs
B **requires** *A*	*A* must occur before *B* occurs	*stem*, *stem-A*, *stem-A-B*
A **forces** *B*	If *A* occurs, then *B* must occur	*stem*, *stem-A-B*, *stem-B*
A **forbids** *B*, or *B* **forbids** *A*	*A* and *B* must not both occur in the same word	*stem*, *stem-A*, *stem-B*

Table 2: Co-occurrence restrictions

INFLECTED feature are "output constraints", so they will be the values of flags to be passed on to other rules.

Table 3 outlines the flag values that must be set to capture the behavior desired. In this table, rules *A* and *B* are assumed to be sequential, and in alphabetical order as before. The "IN" flags are the input constraints, or those on DTR.INFLECTED, while the "OUT" flags are the output constraints, or those on INFLECTED.

When a rule **requires** an earlier rule to have occurred, the earlier rule sets the relevant flag's value to +, and the later rule then requires the input flag's value to be +. If the earlier rule never fires, the flag's value is left as the initial value (*na-or--*), which is not unifiable with +, so the later rule would be blocked.

On the other hand, when a rule **forces** a later rule, it sets the flag value to -, and the later rule would then change it to +. If the later rule never fires, the value is left as -, which is not unifiable with the satisfaction value (*na-or-+)*. In this way, the unsatisfied constraint prevents the lexical object from being used in phrase-structure rules, and thus blocks bad parses.

If a rule **forbids** another rule (regardless of the direction), the earlier rule sets its flag to +, and the later rule requires the input flag's value to be (unifiable with) *na*. If the earlier rule never fires, the value of the flag is the initial value (*na-or--*), which is unifiable with *na* and the satisfaction value.

This configuration of flags for enforcing constraints also works when multiple constraints are applied. For instance, consider a minimal example system with the constraint *B* **requires** *A* and a third sequential rule *C* with the constraint *C* **forbids** *A*. If *A* fires, its flag would be set to +, which satisfies the input constraint on *B*, and is incompatible with the input constraint on *C*. If *A* does not fire, *B* would be blocked, but *C* would then be possible.

Restriction		Flags on *A*		Flags on *B*
B **requires** *A*	IN:		IN:	*a-flag* +
	OUT:	*a-flag* +	OUT:	
A **forces** *B*	IN:		IN:	
	OUT:	*b-flag* -	OUT:	*b-flag* +
A **forbids** *B*, or	IN:		IN:	*a-flag na*
B **forbids** *A*	OUT:	*a-flag* +	OUT:	

Table 3: Flag values for co-occurrence restrictions

Finally, in order to propagate the flag values through chains of lexical rules (of arbitrary length), the system adds constraints to each lexical rule to "copy up" the flag values for any flag not implicated in that rule itself.

4.3.3 Flags, Lexical Rules, and Disjunctions Up until now it might appear that one flag is created for every co-occurrence restriction, but this is not always true. A flag may be shared among multiple co-occurrence restrictions if the restrictions have the same target. Consider a lexical rule *D* with possible inputs *A*, *B*, and *C*. If these three inputs each individually **force** *D*, there would only be one flag (e.g. the *d-flag*) that they all place constraints on. On the other hand, if *D* **required** each of *A*, *B*, and *C*, then three flags would be necessary for the three targets. Also, recall the previous example where the sequential rules *A*, *B*, and *C* have constraints such that *B* **requires** *A* and *C* **forbids** *A*. The constraints are of a different type, but have the same target, and thus only one flag (e.g. the *a-flag*) would be required.

The number of flags can also be reduced by finding disjunctions in the graph of co-occurrence restrictions. There are two kinds of disjunctions: **implicit disjunctions** are when a morphotactic unit has a co-occurrence restriction on two or more nonsequential units, and **explicit disjunctions** are when the user specifies a restriction over a disjoint set of nodes (e.g. "*A* requires *B* or *C*"). The latter case is simple to handle, as it only requires a single flag for the whole set. Implicit disjunctions are more difficult to handle. Consider the input graph in Figure 15. *B* and *D* both take *A* as input, *C* takes *B* as input, and *E* takes *D* as input. Now assume that *A* requires, individually, each of *B*, *C*, *D*, and *E*. Having four flags (one for each) would not only be inefficient, but incorrect, since applying rules on either path would preclude the satisfaction of flags on the other path. We could choose the first rule on a path and make a disjoint set with each rule on the other path until all nodes are covered (e.g. *B-or-D*, *B-or-E*, *C-or-D*), but we end up with three flags, which is still not ideal. If, instead, we keep track of which restrictions have been accounted for, and select the next available restriction for the creation of new flags, we end up with two flags: *B-or-D*, and *C-or-E*.

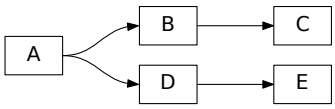

Figure 15: An implicit disjunction of co-occurrence restrictions

4.3.4 Obligatoriness A fundamental part of describing a morphotactic system is specifying which position classes are obligatory and which are optional. O'Hara (2008)'s system had the user directly specify whether a position class was optional or obligatory. This was crucial to her system's logic for modeling morphotactics. In my system, obligatoriness simply means the position class is required by its inputs, and is thus modeled entirely with flags. In this view, a position class being required by a lex-type is functionally no different than it being required by some other position class. I, however, continue to allow the user to specify "obligatoriness" as a convenience. When a

user specifies that a position class is obligatory, it is interpreted as having every lex-type (i.e. "root") that can lead to the position class **force** it. The user could manually specify these restrictions, and the output grammar would be identical to one had they said the position class was obligatory. Decomposing obligatoriness to individual constraints also allows restrictions to be placed on smaller sets of lexical items (that is, the position class would obligatory for only a subset of the lexical types and be optional for others).

5 Evaluation and Test Cases

This new morphological infrastructure has an increased power to model morphotactic systems over O'Hara's previous infrastructure, and this section illustrates the differences with some evaluative metrics and concrete examples. The major improvements are the complex (and non-binary) conditions for inflectional satisfaction, disjunctive co-occurrence restrictions, and the ability for a user to define lexical rule type hierarchies (in position classes). I explain how the changes affect parsing performance in Section 5.1, while at the same time showing that my system could model the languages O'Hara's system was evaluated against. I then show how the new features help model morphological systems that were previously impossible to model elegantly with examples from Lushootseed in Section 5.2, French in Section 5.3, and Chintang in Section 5.4. Since one of my goals is to make the human effort of developing a grammar easier, in Section 5.5, I consider the change in complexity in both the questionnaire and the resulting grammar.

5.1 Parsing Performance

I tested the parsing performance[18] of the Grammar Matrix customization system just before and just after implementing the new morphotactic framework.[19] I tested against several of the languages O'Hara used to evaluate her system, and several others in addition. Figure 16 shows the difference in the number of active edges, passive edges, and unifications used in parsing. The edges and unifications are simply metrics reported by the parser that explain how much space or work was utilized in producing a parse. Negative numbers indicate the newer system used fewer edges/unifications, and was thus more efficient, whereas positive numbers show a degradation in performance.

For the most part, performance is identical, which is a welcome result. The new framework is able to model more phenomena without harming performance. Two languages, Cree and Fore, showed an increase in performance (with the latter having 45 fewer unifications on average). These two languages are the only ones in the set that exhibit a direct-inverse case-marking system (see Drellishak, 2009). Many lexical rules used in the direct-inverse grammars are implicit (i.e. not-specified by the user), so this result suggests that my system creates more efficient models of the phenomenon.

The only language with a degradation in performance is Slave, using 1 more passive edge and 16 more unifications on average. This western-Canadian Athabaskan language is highly synthetic

[18] To be clear, by "performance" I mean the computational complexity (time and space used) in parsing sentences, and not "competence", or the coverage of the system on the range of grammatical and ungrammatical sentences. The new system was able to accept and reject the same sentences the previous system could.

[19] These correspond to Subversion revision numbers 15354 and 15448, respectively.

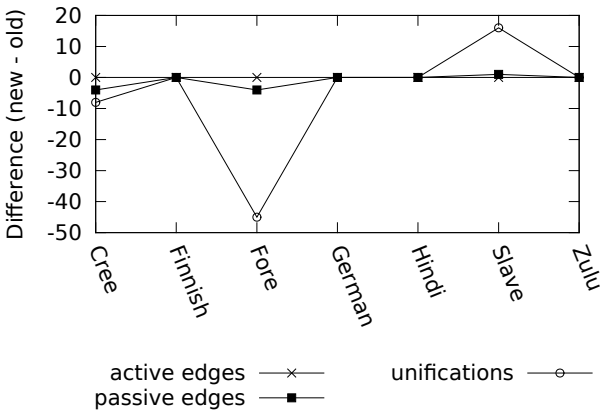

Figure 16: Parsing performance before and after the new framework (lower numbers show improvement)

and uses co-occurrence restrictions more heavily than the other languages tested. While the new system is clearly superior in its ability to model morphological systems, the old system appears to be more efficient when multiple co-occurrence restrictions are used. This is likely due to the old system's pruning of implicit inputs based on the obligatoriness of position classes, whereas the new system models it entirely with flags. Unfortunately, the fact that position classes may now contain lexical rule hierarchies instead of just single rules makes this pruning step much more difficult, and thus this is a potential line of future research.

5.2 Lushootseed

Lushootseed [lut] is an agglutinative Coast Salish language. It has affixes for tense and for aspect, with the requirement that at least one of them (tense or aspect) must occur, or both may occur (Hess, 1967). (1) is an example of a sentence with both tense and aspect specified. With the old system, this disjunctive relationship would require complex arrangements, and much duplication of position classes and rules, but with the new system it can be modeled more elegantly.

(1) Lulexwil ti cacas
Lu-le-xwil t-i cacas
FUT-PRG-get.lost det.DEF-DIR child
'The child will become lost.' [lut]

The user would first create the position classes and set their inputs as needed to capture the affix ordering correctly, then add a disjunctive "forces" restriction on all lexical types that require a tense or aspect marker, with the forcees being the tense and aspect position classes. The system will create one flag for these two position classes, and when one position class occurs the flag is satisfied. If both occur, the value of the flag is still satisfied, but if neither occur it will be left unsatisfied, disallowing the lexical object from being used in phrasal rules. Note that the Lushootseed case as shown in Figure 17 differs from that depicted in Figure 15, in that the disjunctively forced position

classes are on the same input path (i.e. they are sequential), allowing them to co-occur. While this configuration is very simple to model in the new system, O'Hara (2008)'s system would require at least one duplicated position class.[20]

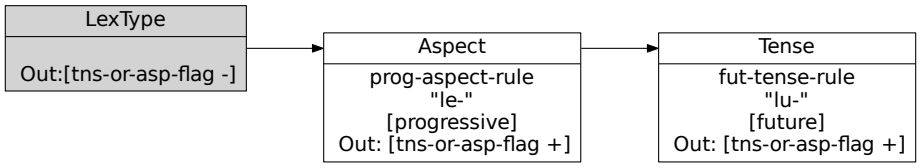

Figure 17: Lushootseed sequential disjunctive requirements

5.3 French Pronominal Affixes (or Clitics)

In French [fra], strict transitive verbs such as *prendre* ('take') require either object clitic (analyzed as a prefix, following Miller and Sag (1997)) or a full NP object, while optionally transitive verbs such as *manger* ('eat') can take the clitic, a full NP object, or no object at all:[21]

(2) Je mange le biscuit
I eat.1SG the cookie
'I eat the cookie.' [fra]

(3) Je le-mange
I eat.1SG 3SG.M-eat1SG
'I eat it/I 3SG.M-eat1SG.' [fra]

(4) Je mange
I eat.1SG
'I eat ϕ.' [fra]

(5) Je prends le biscuit
I take.1SG the cookie
'I take the cookie.' [fra]

(6) Je le-prends
I 3SG.M-take1SG
'I take it.' [fra]

[20] One complication of O'Hara (2008)'s system that has not been mentioned is that there are lexical rule types for changing INFLECTED's value from $-$ to $+$, and they not only affect the output value, but check that the input value is the opposite. Thus, using those rule types, it is impossible to "reapply" a $+$ value to $+$.

[21] We assume for the sake of the example that there is only one lexical item for *manger*.

(7) *Je prends
I take.1SG
'*I take ϕ.' [fra]

The argument optionality library in the Grammar Matrix (Saleem, 2010) handles such alternations via lexical rules. For a verb like *prendre*, there is an obligatory position class which houses the object prefix rules as well as a zero-marked rule which constrains the object to be incompatible with the object-drop phrase structure rule. A verb like *manger*, however, should not go through this rule. Rather, it should simply optionally take the object prefixes. With the old system, this required duplicating a position class. With the new system, however, it can be elegantly handled by placing a "forbids" restriction on the lexical type for *manger* so it cannot take the zero-marked rule and a "requires" restriction on *prendre* so that it must either take an object prefix or the zero-marked rule. This configuration is illustrated in Figure 18.

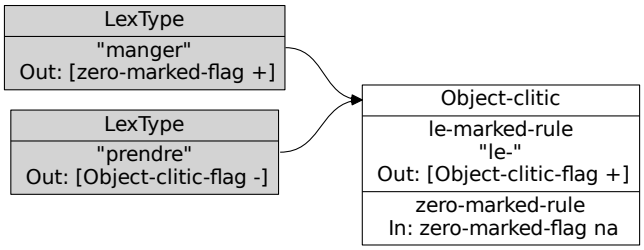

Figure 18: French object clitics

5.4 Chintang Bipartite Stems and Free Prefix Ordering

Chintang [ctn] is an Eastern Kiranti language of Nepal and is noted for its free prefix ordering and bipartite stems. Here I will explain how bipartite stems can be modeled and discuss how free prefix ordering could be handled with an addition to the framework. All examples are from Bickel et al. 2007.

(8) kha-u-kha-ma-siŋ-yokt-e
ask-3NS.A-1NS.P-NEG-ask-NEG-PST
'They didn't ask us.' [ctn]

(8) gives an example of bipartite stems. The two parts of the stem for "ask" are *kha* and *siŋ*. They cannot occur independently, and together they contribute one meaning ("ask"), but they act as separate morphological units. The bipartite stems can be modeled as though one is a semantically empty lexrule required by the "primary" stem. For instance, the user can create a root *kha*, and then have it require an affix that does nothing but add the orthography *siŋ*. Because the most specific place a user may target a co-occurrence restriction is a lexical rule type, the user would have to

create a lexical rule type for every bipartite stem. This superfluence of types could become very large, particularly if bipartite stems are prevalent in the lexicon. To alleviate this issue, users are allowed to specify bipartite stems directly when creating the lexicon (as opposed to regular stems), and during customization the system will create co-occurrence restrictions targeting the lexical rule instances, rather than types.[22]

Chintang's free prefix ordering can be seen in (9) and (10). Both sentences have identical semantics, but differ on the order of the prefixes on *im* "sleep". The user could create position classes that take each other as input, but this runs the risk of rule-spinning since a lexrule's constraints would unify with themselves. A solution to this problem is that a lexrule could forbid itself (the input would check the INFLECTED flag value for *na*, and the output would set it to +) thus only allowing the rule to occur once in a word. This arrangment can be seen in Figure 19. Note that this would work even if the lexrules are obligatory (e.g. the lextype specifies the flags as -). I do not allow the user to specify such a configuration in the questionnaire because cycles in the input graph may cause infinite loops in processing, therefore I check for and prevent the user from creating cycles of inputs. The user may bypass this restriction and enter the configuration manually during post-editing (see, for example, Bender et al., 2012a). Allowing cyclic input graphs is a task deferred to future work.

(9) a-ma-im-yokt-e
2-NEG-sleep-NEG-PST
'You didn't sleep.' [ctn]

(10) ma-a-im-yokt-e
NEG-2-sleep-NEG-PST
'You didn't sleep.' [ctn]

5.5 Ease of Human Effort

The changes I introduced resulted in changes to both the Grammar Matrix questionnaire and the resulting grammar, and in some ways defining a morphological system is more complex, while in others it is more convenient. There are tradeoffs with notational complexity versus expressive power, but I don't think my changes result in a more difficult development experience.

In O'Hara's system, the user defined lexical rules in the questionnaire individually, so for each one the user had to specify the rule inputs, feature constraints, and co-occurrence restrictions. Now the user must define three levels (position class, lexical rule type, and lexical rule instance), but information may be placed higher up in the hierarchy and inherited, thus reducing redundancy. While the system is capable of using the same co-occurrence restrictions (require previous, force following, and forbid co-occurrence), I simplified the questionnaire so the user is presented with only two choices: require and forbid. I do this because the user does not need to be aware of the implementational distinction while filling out the questionnaire, and also to prevent illogical situations such as

[22] Outside of bipartite stems, I can't think of an example that is typologically common enough to necessitate allowing the user to specify co-occurrence restrictions on lexical rule instances, so I do not generally make the faculty available. Users can, of course, make them manually in post-customization editing.

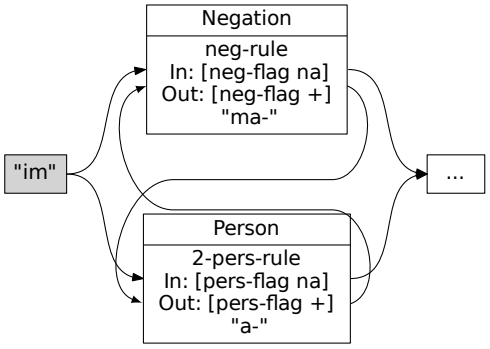

Figure 19: A solution for free prefix ordering in Chintang

requiring the previous occurrence of a rule that wouldn't occur until later. Because co-occurrence restrictions may be defined on lexical types, position classes, or lexical rule types, there is some amount of "clutter" in the questionnaire when the user makes no use of these restrictions, but I would argue the additional buttons are worth the increase in expressive power.

In the resulting grammar, the lexical rule hierarchies are similar to other type hierarchies in the grammar, so they should match developer's expectations as compared to a listing of lexical rules that may redundantly specify constraints. In O'Hara's system, there were two features for modeling inflectional satisfaction: INFLECTED and the matrix of flags (called TRACK in her system), but in my system they are unified. The one place my system falls short is in the verbosity of setting, checking, and copying flag values. The previous system pruned rule inputs, when possible, to ensure obligatory rules occurred, but my system uses flags to accomplish this, resulting in more flags than are strictly necessary (although one might argue the flags make the obligatoriness more transparent; and further, allow finer-grained control over which lex-types obligatorily take the rules). Moreover, when multiple co-occurrence restrictions are used, each lexical rule may have more lines of code devoted to flag management (e.g. copying up unmodified flags) than the syntactic or semantic constraints introduced by the rule. My system, however, does not significantly differ from O'Hara's system in this regard.

6 Related Work

Now that I've described my system, I will compare it to other approaches to computational morphology. I will begin in Section 6.1 with a very brief overview of the early methods, such as finite-state transducers and two-level automata. In Sections 6.2 and 6.3 I discuss flag diacritics and feature-driven morphotactics—two methods that bear resemblance to the co-occurrence restrictions used in my system.

6.1 Early Methods of Computational Morphology

By the 1980s, formal morphology had on one side the Chomskyan generative phonologists with their ordered rewrite rules, and on the other side computational linguists with simple "cut-and-paste" systems for analysis. While the former could generate and the latter could analyze, neither were readily applicable to the other task.[23]

It was recognized that finite-state transducers (FSTs) could be applied to the task, but while they are inherently reversible, an FST that is efficient in generation is not necessarily efficient in analysis. For example, consider a simple monodirectional rule $X \rightarrow Y$. In generation there is a one-to-one mapping from X to Y, but in analysis, given a Y the underlying string is ambiguous between X and Y. This ambiguity becomes unwieldy for any significant number of rules. A solution is to compose the lexicon with the rules into a single transducer so only valid lexical items are analyzed, but this idea wouldn't come until later.

In 1981, Kimmo Koskenniemi had an idea for morphological rule framework that could be used for generation and analysis (i.e. reversible rules). His method was called **Two-Level Morphology** (Koskenniemi, 1984) because its rules could only constrain the lexical and/or surface forms; there were no intermediate representations from cascading sets of rules. In order to allow for multiple independent rules applying on the same word, all rules (or rather, the intersection of their constraints) are applied in parallel, rather than sequentially. Rules could be conditioned on the context of either, or both, the original and output forms. While two-level rules don't solve the overanalysis problem, their avoidance of intermediate forms makes the computation more tractable.

As an example, Figure 20 shows two rules that are applied in parallel to model the $y \rightarrow ie$ alternation in forming the plural *spies* from the singular *spy*. The first rule is read as: "*y* becomes *i* only when it precedes an epenthetic *e*". The colon `:` signifies the division of the two levels, so the *0* to the left of the colon is a lexical *0* and the *e* to the right of the colon is a surface *e*.[24] A character without a colon refers only to the lexical form and a character preceded only by a colon (e.g. `:c`) refers only to the surface form. The underscore `_` represents the segment affected by the rule. The second rule is read as: "always insert an *e* when it follows a lexical consonant-*y* cluster and precedes a morpheme boundary (+) and a lexical *s*".

```
y:i => _ 0:e     0:e <= C y _ + s
```

Figure 20: Two-level rules used in converting *spy* to *spies*

Two-level morphology describes the orthographic changes from a lexical form to a surface form. What it doesn't specify is the relationship of words to the morphemes they can take. In the example above, in order to get *spies* from *spy*, the input to the rules must be of the form `spy+s`, which is the lexeme SPY with a morpheme boundary and the s morpheme. In analysis the word *spies* would be mapped back to `spy+s`. The formalism does not specify the valid combinations or orderings

[23] For more information, Karttunen and Beesley 2005 offers an excellent and concise history of the field.

[24] There is a requirement that both sides of a rule must have the same number of characters, so any inserted or deleted characters must have an explicit epsilon (*0*) counterpart. These epsilons are not part of the word in the lexicon, but all rules with constraints involving epsilons must be consistent in their number and location.

of morphemes and lexemes, except insofar as they match defined orthographic rules, nor can it handle long-distance constraints. Furthermore, it is not explicitly modeling the morphosyntactic and morphosemantic effect of the affixes.

6.2 Flag Diacritics

To address long-distance constraints, Beesley and Karttunen (2003) introduced **flag diacritics**. In form they are merely multicharacter symbols that appear in the standard two-level or finite-state rules, and are treated like epsilons[25] on input and output. In function, however, they are used to set[26] and test a range of user-defined features and values at runtime, thus preventing the generation or analysis of incompatible morphemes. As the flag diacritics are simple multicharacter symbols, they are given an interpretable internal structure to allow complex restrictions. The general pattern is `@Op.Feat.Val@`, where the entire flag is surrounded by `@` characters and the subcomponents are delimited by period (`.`) characters. `Op` is the operator being used to set or test the feature `Feat` with the value `Val`. While there are a predefined set of operators, the feature and value strings are to be decided by the user. Table 4 describes the different operators available to the user. For most cases, though, the Unification test is sufficient.

Synopsis	Purpose	Notes
`@P.Feat.Val@`	Set Positive	Sets `Feat` to `Val`. Never causes failure.
`@N.Feat.Val@`	Set Negative	Sets `Feat` to ¬`Val`. Never causes failure.
`@R.Feat.Val@`	Require	Succeeds if `Feat` is already set to `Val`, otherwise fails.
`@R.Feat@`	Require Any	Succeeds if `Feat` is set to any value, including negated values, otherwise fails.
`@D.Feat.Val@`	Disallow	Succeeds if `Feat` is unset or set to any value other than `Val`, otherwise fails. Also succeeds if `Val` was negated, and fails if any other value was negated. For instance, the sequence `@N.F.X@@D.F.Y@` fails, while `@N.F.X@@D.F.X@` succeeds.
`@D.Feat@`	Disallow Any	Succeeds only if `Feat` is unset, otherwise fails.
`@C.Feat@`	Clear Feature	Unsets `Feat`, clearing any positive or negative value settings.
`@U.Feat.Val@`	Unification	Succeeds if `Feat` is currently set to `Val` or the negation of any other value or is unset. If `Feat` was unset or negated, it is set to `Val`.

Table 4: Operations on flag diacritics

I was unaware of flag diacritics when I created the work described in this paper, and there are a number of similarities and differences between the two approaches. Flag diacritics have a larger

[25] In finite state theory, an epsilon represents the empty string.

[26] The ability to store feature-value state in memory means the formalism no longer represents strict FSTs. For flag-unaware applications (i.e. traditional FSTs), the flag diacritics can easily be ignored, but the flagless system will likely overgenerate.

range of operators, but my approach appears to cover all of their functionality (although recreating the negated feature values can be cumbersome). My approach, however, also allows feature hierarchies instead of a flat structure. For instance, if `1SG` is a subtype of `non-3SG`, a rule could specify the former and another rule would succeed in requiring the latter (since `1SG` is a kind of `non-3SG`), but with flag diacritics the requirement check would not succeed. While flag diacritics can be used with cascading rules or two-level rules, my system is only designed to work with cascading rules. My system does not have explicit support for clearing flag values, but this can be accomplished[27] by not copying flag values from the input to the output. My system also has support for disjunctive as well as conjunctive flag combinations, where flag diacritics only support conjunctive.

6.3 Feature-Driven Morphotactics

Another work I was unaware of when I began was that of Hulden and Bischoff (2007). Hulden and Bischoff devise a formalism[28] for constraining morpheme co-occurrence that gets away from the concatenative, continuation-class models of morphotactics so it can better support phenomena like free morpheme order, circumfixation, etc. They separate the specification of morpheme co-occurrence restrictions from the specification of morpheme-order, which could even be left out entirely to allow for free morpheme-order. They define the four morpheme-order operators given in Table 5.

$C_1 < C_2$	A morpheme of class C_1 must immediately precede a morpheme of class C_2.
$C_1 << C_2$	A morpheme of class C_1 must precede (not necessarily immediately) a morpheme of class C_2.
$C_1 > C_2$	A morpheme of class C_1 must immediately follow a morpheme of class C_2.
$C_1 >> C_2$	A morpheme of class C_1 must follow (not necessarily immediately) a morpheme of class C_2.

Table 5: Morpheme-order Restrictions defined by Hulden and Bischoff (2007)

Hulden and Bischoff offer a partial analysis of Chintang's free prefix ordering using their morpheme-ordering formalism. As shown in Figure 21, prefixes in classes $C_1 \ldots C_n$ with free ordering merely have the constraint that they precede morphemes of class C_x, while morphemes $C_x \ldots C_y$ have a strict ordering.

$$\begin{gathered} C_1 << C_x \\ \cdots \\ C_n << C_x \\ C_x << \cdots << C_y \end{gathered}$$

Figure 21: Modeling Chintang's free prefix ordering

[27] By hand development, using the structures provided by my system. This is not possible through the customization system.

[28] While they aspire to be implementation-neutral, they provide an FST-based implementation as a proof-of-concept.

Unlike flag diacritics, Hulden and Bischoff only define three operators for co-occurrence restrictions, given in Table 6. My approach has similar co-occurrence restrictions, and also models morpheme order separately, but unlike their approach, in my system co-occurrence restrictions have cascading effects. In other words, the consequences of earlier rules affect rules occurring later. Also, while the interface to my system does not explicitly allow free morpheme-order, nothing in the formalism prevents it, so it is possible with a simple extension.

$\sqcup\|F\ X\|$	Unify	Fails if $\sqcup\|F\ Y\|$ appears in the same word for any $Y \neq X$.
$+\|F\ X\|$	Coerce	Fails if $\sqcup\|F\ X\|$ does not appear in the same word.
$-\|F\ X\|$	Exclude	Fails if $\sqcup\|F\ X\|$ appears in the same word.

Table 6: Co-occurence Restrictions defined by Hulden and Bischoff (2007)

7 Conclusion

This paper described a system that provides a structured way to elicit information about a morphological system from a user and then creates a model of that system in a machine-readable grammar. It expands on the system of O'Hara (2008) by allowing features for co-occurrence restrictions to appear directly on the INFLECTED attribute. This change helps to ensure that inflectionally unsatisfied lexical items are unable to be used with phrase structure rules, particularly for complex co-occurrence restrictions where a boolean distinction for inflectional satisfaction was insufficient. Furthermore, this change allowed the elegant description of complex morphological models where previously such descriptions were impossible.

The user can now place lexical rules types in hierarchies, which reduces redundancy when syntactic or semantic properties cross-classify over a set of lexical rules, and allows for restrictions over generalized sets of rules. Co-occurrence restrictions can now be placed on any node (lexical type, lexical item, position class, lexical rule type, lexical rule instance) available to the morphotactics component, and any of these nodes can be input to a position class. Explicit and implicit support for disjunctive inputs and co-occurrence restrictions allowed for concise definitions of "flexible" morphologies (where a lexical object may take multiple paths to inflectional satisfaction).

On the surface, these changes may seem small, but in making them the expressive power of this morphology component of the Grammar Matrix customization system is greatly expanded. For example, the support for disjunctive co-occurrence restrictions allows one to describe systems, such as with the Lushootseed language, where either one affix or another are required, but also both may to occur in the same word. Without the mechanisms provided by this work, the system could only be modeled by creating duplicate lexical rules.

For the resulting grammars, the system also reduces inefficiency in type and attribute declarations, thus making them easier for humans to read and understand, aiding in subsequent development. Grouping rules with the same possible inputs into position classes makes it relatively painless to later add an intervening rule or position class between the original position class and its input. Arranging the rules into lexical rule hierarchies allows the developer to place co-occurrence restrictions or syntactic/semantic constraints higher up in the hierarchy and thus cover a class of rules, rather than duplicating these restrictions or constraints on all sub-rules.

However, the effects of this system aren't all positive. I haven't yet written code to model co-occurrence restrictions by pruning inputs where possible, and this results in the creation of some unnecessary flags. Moreover, any flags whose values are not set in a rule must be explicitly copied in the lexical rules. From a parsing-performance standpoint, more computational work is required to process lexical rules where multiple co-occurrence restrictions are employed.

Further, there are still morphophonological patterns that are not easily modeled by my system, but these are arguably hindered by limitations in the DELPH-IN formalisms, and not a property of my system. For example, circumfixes cannot be modeled by a single lexical rule, and instead must be defined with multiple rules that require each other (so they are guaranteed to co-occur). Bipartite stems are given a similar treatment to circumfixes. Infixes, as one may use to describe Arabic morphology such as the inflection of *kitab* "book" and *kutub* "books" from the lexeme *ktb* BOOK, cannot be defined unless the various parts of the root are instead treated as affixes. There is a mechanism for simple base modification in LKB-style grammars, but I don't give users access to this mechanism in the questionnaire (they can always write rules with it in subsequent development). I'm not too concerned with these shortcomings as the goal of my system is to model morphotactics, and simple affixation is enough to show the order of rule application. For example, I would suggest an Arabic grammar to output something like KTB-PL, and an external morphological processor can rewrite its form to *kutub* "books". My system is rather used to ensure things like the over-inflected KTB-SG-PL or under-inflected KTB are not used in phrasal rules.

The code for this morphotactics framework is embedded in the Grammar Matrix project and is freely available under the MIT license.[29] Instructions for obtaining the code are at `http://www.delph-in.net/matrix/`. The Grammar Matrix questionnaire can be used at `www.delph-in.net/matrix/customize/`.

References

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publications.

Emily M. Bender, Scott Drellishak, Antske Fokkens, Michael Wayne Goodman, Daniel P. Mills, Laurie Poulson, and Safiyyah Saleem. 2010a. Grammar Prototyping and Testing with the LinGO Grammar Matrix Customization System. In *Proceedings of ACL 2010 Software Demonstrations*.

Emily M. Bender, Scott Drellishak, Antske Fokkens, Laurie Poulson, and Safiyyah Saleem. 2010b. Grammar Customization. *Research on Language & Computation*, 8(1):23–72. 10.1007/s11168-010-9070-1.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In *Proceedings of the Workshop on Grammar Engineering and Evaluation at COLING 2002*, pages 8–14.

[29] `http://opensource.org/licenses/MIT`

Emily M. Bender and Jeff Good. 2005. Implementation for Discovery: A Bipartite Lexicon to Support Morphological and Syntactic Analysis. In *Proceedings from the Panels of the Forty-First Meeting of the Chicago Linguistic Society: Volume 41-2.*, pages 1–15.

Emily M. Bender, Robert Schikowski, and Balthasar Bickel. 2012a. Deriving a Lexicon for a Precision Grammar from Language Documentation Resources: A Case Study of Chintang. In *Proceedings of the 24th International Conference on Computational Linguistics*. Association for Computational Linguistics.

Emily M. Bender, David Wax, and Michael Wayne Goodman. 2012b. From IGT to Precision Grammar: French Verbal Morphology. In *LSA Annual Meeting Extended Abstracts*.

Balthasar Bickel, Goma Banjade, Martin Gaenszle, Elena Lieven, Netra Paudyal, Ichchha Purna Rai, Manoj Rai, Novel Kishor Rai, and Sabine Stoll. 2007. Free Prefix Ordering in Chintang. *Language*, 83.

Joan Bresnan and Sam A. Mchombo. 1995. The Lexical Integrity Principle: Evidence from Bantu. *Natural Language & Linguistic Theory*, 13(2):181–254.

Joan L. Bybee. 1985. *Morphology: A study of the relation between meaning and form*, volume 9. John Benjamins Publishing Company.

Ann Copestake. 2002a. Definitions of Typed Feature Structures. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*, pages 227–230. CSLI Publications, Stanford, CA.

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

Ann Copestake, Alex Lascarides, and Dan Flickinger. 2001. An Algebra for Semantic Construction in Constraint-based Grammars. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 140–147. Association for Computational Linguistics.

B. Crysmann. 1999. Morphosyntactic paradoxa in Fox. In *Joint Conference on Formal Grammar, Head-Driven Phrase Structure Grammar, and Categorial Grammar*, page 247. Citeseer.

Scott Drellishak. 2009. *Widespread But Not Universal: Improving the Typological Coverage of the Grammar Matrix*. Ph.D. thesis, University of Washington.

Dan Flickinger. 2000. On Building a More Efficient Grammar by Exploiting Types. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):15 – 28.

Martin Haspelmath and Andrea D. Sims. 2010. *Understanding Morphology*. Understanding Language Series. Taylor & Francis Limited.

Thomas M. Hess. 1967. *Snohomish grammatical structure.* Ph.D. thesis, University of Washington.

Mans Hulden and Shannon T. Bischoff. 2007. A Simple Formalism for Capturing Order and Co-occurrence in Computational Morphology. *Procesamiento del Lenguaje Natural*, 39:21–26.

Lauri Karttunen and Kenneth R Beesley. 2005. Twenty-five Years of Finite-state Morphology. *Inquiries Into Words; a Festschrift for Kimmo Koskenniemi on his 60th Birthday*, pages 71–83.

Kimmo Koskenniemi. 1984. A General Computational Model for Word-Form Recognition and Production. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 178–181. Association for Computational Linguistics.

Christopher Manning, Ivan A. Sag, and Masayo Iida. 1999. The lexical integrity of Japanese causatives. *Studies in contemporary phrase structure grammar*, pages 39–79.

Philip H. Miller and Ivan A. Sag. 1997. French Clitic Movement without Clitics or Movement. *Natural Language and Linguistic Theory*, 15:573–639.

Kelly O'Hara. 2008. *A Morphotactic Infrastructure for a Grammar Customization System*. Master's thesis, University of Washington.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Safiyyah Saleem. 2010. *Argument Optionality: A New Library for the Grammar Matrix Customization System*. Master's thesis, University of Washington.

Gregory T. Stump. 2001. *Inflectional morphology: A theory of paradigm structure*. Cambridge Univ Press.