A System for Archivable Grammar Documentation

Michael Maxwell

University of Maryland, College Park MD 20742, USA
mmaxwell@umd.edu

Abstract. This paper describes a number of criteria for archivable documentation of grammars of natural languages, extending the work of Bird and Simons' "Seven dimensions of portability for language documentation and description." We then describe a system for writing and testing morphological and phonological grammars of languages, a system which satisfies most of these criteria (where it does not, we discuss plans to extend the system).

The core of this system is based on an XML schema which allows grammars to be written in a stable and linguistically-based formalism, a formalism which is independent of any particular parsing engine. This core system also includes a converter program, analogous to a programming language compiler, which translates grammars written in this format, plus a dictionary, into the programming language of a suitable parsing engine (currently the Stuttgart Finite State Tools). The paper describes some of the decisions which went into the design of the formalism; for example, the decision to aim for observational adequacy, rather than descriptive adequacy. We draw out the implications of this decision in several areas, particularly in the treatment of morphological reduplication.

We have used this system to produce formal grammars of Bangla, Urdu, Pashto, and Persian (Farsi), and we have derived parsers from those formal grammars. In the future we expect to implement similar grammars of other languages, including Dhivehi, Swahili, and Somali. In further work (briefly described in this paper), we have embedded formal grammars produced in this core system into traditional descriptive grammars of several of these languages. These descriptive grammars serve to document the formal grammars, and also provide automatically extractable test cases for the parser.

1 Introduction

I will take it as given that one of the goals in language documentation is to create descriptions which will be usable as long as possible—preferably for hundreds, if not thousands, of years. This paper discusses design criteria for computer-supported morphological analysis in support of that goal.

The earliest morphological parsers were written in ordinary programming languages for specific languages. Hankamer's *keçi* [10], for example, was written in the C programming language, primarily to parse Turkish.[1] One implication of this sort of design is software obsolesence; the knowledge about the grammar is encoded in a format

[1] Hankamer suggests that the program might be useful to parse other agglutinating languages, but as far as I can determine, it was never used in that way.

C. Mahlow and M. Piotrowski (Eds.): SFCM 2013, CCIS 380, pp. 72–91, 2013.

(the C programming language) which is destined to some day be obsolete. Moreover, the language-specific linguistic aspects of the analysis are intermingled with language-general aspects (what a suffix is, for example), not to mention information which has nothing to do with linguistics, such as the way for-loops are encoded in the C programming language.

The use of language-independent parsing engines for morphological parsing was the first step beyond programs designed from the ground up for a particular language. Programs like AMPLE (developed by SIL in the 1980s: [26]) and the Xerox XFST finite state transducer [1], along with many other such tools provide language-general knowledge about many aspects of morphology and phonology. AMPLE, for example, allows one to build a database of allomorphs conditioned by phonological environments. XFST allows in addition the statement of various kinds of phonological alternations or processes, which can generate allomorphs from underlying forms.

Such language-independent parsing engines represent an important step towards linguistically motivated computational descriptions, in that they release the writer of a language description from the necessity of building programming tools to treat morphotactics, phonotactics, phonological environments, and phonological processes. This is a necessary step, but not a sufficient one. A further step is needed to ensure the longevity of those linguistic descriptions, so that they may be consulted by future generations of linguists, and used for morphological parsing long after any particular parsing engine is obsolete and unusable. This paper describes a framework with those characteristics: it supplies a linguistically based notation, one which will be familiar to most linguists; and the notation, being stated in XML, is stable and (as much as such a notation can be) self-documenting.

2 Criteria for Grammatical Descriptions

In this section I motivate criteria for archivable grammatical descriptions, and use these criteria to argue for a way of describing grammars which is computationally implementable and at the same time indepenent of any particular parser implementation. I begin with a seminal document in the language description literature, *Seven dimensions of portability for language documentation and description*.

2.1 Seven Pillars for Language Description

Bird and Simons [2] discuss the requirements for producing archivable descriptions of languages.[2] Among these requirements are the following:[3]

[2] Bird and Simons make a distinction between "documentation," that is primary language data such as recordings and transcriptions, as opposed to "description," that is a linguistic analysis of the primary language data. Since these terms are easily confused, I will use "description" in the same sense as Bird and Simons, to refer to the linguistic analysis, but "data" to refer to what Bird and Simons call documentation.

[3] I omit some of Bird and Simons' criteria which seem less relevant to the discussion here, namely *discovery, access, citation, and rights*.

1. Content: Among content-based requirements, Bird and Simons include:
   Accountability: By "accountability," they mean the ability to verify the description against actual language data.
   Terminology: The terminology used in a language's description should be defined, e.g., by pointing to standardized ontologies.
2. Format: This refers to the structure of the file in which the description is housed.
   Openness: Linguistic descriptions should use formats which conform to open standards.
   Encoding: Unicode is preferable.
   Markup: Bird and Simons call for plain text markup formats. They further argue that the markup should be descriptive, not presentational, with XML as the standard for such markup.
   Rendering: There needs to be a method to render linguistic documents in human-readable form.
3. Preservation: This desideratum refers to the need for linguistic descriptions to be archivable, as well as to be archived. Bird and Simons emphasize the need for *longevity*, that is, planning for the use of the resource for periods of decades (or, one may hope, centuries).

2.2 More on Pillars for Grammatical Descriptions

Bird and Simons are not explicit about the sorts of language descriptions for which these criteria are relevant, but they discuss textual descriptions of languages, annotated corpora, and lexical resources. It may not be at first glance clear how the above criteria apply to grammatical resources, and specifically to morphological descriptions. I will therefore elucidate in this section specific ways in which grammatical resources must be created if they are to support the general goals of *content, format*, and *preservation* which Bird and Simons outline. I will also discuss some additional criteria which are more specific to grammatical descriptions.

To begin, in order for a grammatical description to meet their criterion of accountability—the ability to validate a description against primary language data—it must be possible to test such a description on actual language data. While it may seem that this can be done by pure thinking (aided, perhaps, by pencil and paper), the last decades of computational linguistics have shown nothing if they have not demonstrated that grammatical descriptions are hard to debug. While this has long been clear in syntax, it has also become clear in morphology and phonology (e.g., [3,16,27,28]). These last two references describe problems arising in the interpretation of Newman's [22] description of Yokuts (= Yawelmani, or Yowlumne). As Weigel [27,28]) and Blevins [3] make clear, the misinterpretations have resulted in something of a disaster for theoretical phonology, in that fundamental claims in generative phonology turn out to have been supported by misunderstandings of Newman's work.

It is not the case that Newman's description was unintelligible or inherently faulty; no less a linguist than Zellig [12, p. 196] described it in glowing terms:

> Newman's long-awaited Yokuts grammar is [...] a model contribution to descriptive linguistic method and data. It is written clearly and to the point, in

> a matter that is aesthetically elegant as well as scientifically satisfactory. It is sufficiently detailed [...] to enable the reader to become familiar with the language and to construct correctly his own statements about the language. Phonology and morphology are treated fully [...] students and workers in linguistics should read [this] with close attention to the method of handling descriptive and comparative data.

Nevertheless, Weigel [28] writes:

> Newman's explanations and descriptive rules of Yokuts morphology are often not completely clear. Indeed, no less a linguist than Charles Hockett had to admit (in Hockett 1973) that he had misapplied some of Newman' s rules in an earlier published piece (Hockett 1967).

One might ask why linguists mis-construe the output of grammars. Certainly complexity is one aspect; for any non-trivial grammar, it is difficult to think through all the implications of all the rules on all the lexical items and affixes. But there is another reason. One may view a grammar as the description of a piece of software. In this case, the "software" is originally implemented as "wetware," that is in people's brains; the task is to describe that program (or at least generate its outputs) clearly and unambiguously.[4] The problem of grammatical description is thus an issue of software documentation. And as is well known, verbal descriptions of software are inherently and nearly unavoidably ambiguous. Thus, in addition to the complexity of natural language grammars themselves, we have the ambiguity of their descriptions.

If then a model grammar such as Newman's can be so misinterpreted, what hope is there for the average grammar? The hope, I contend, is that we should use computers to help us validate and understand grammars. But it is obvious that computers cannot interpret descriptive grammars, written in English or any other natural language—computers are actually worse at this task than humans are. We therefore require computationally implementable and testable grammars. By implementing such a grammar, we can arrive at a description which can unambiguously answer the questions we put to it, such as "What is the complete paradigm of verb X?", a question which (as Weigel notes) is difficult to answer from Newman's description.

We thus arrive at the first of several criteria for adequate morphological descriptions:

Criterion 1. *A morphological (or more generally, grammatical) description must be computationally implementable.*

This criterion in support of Bird and Simon's pillar of *accountability* immediately raises questions. In particular, what description language should we use? Obvious candidates are the programming languages used by modern morphological parsing engines. But the problem with this answer should be clear from the plural suffix on "candidates":

[4] It is possibile that mental grammars are inherently "fuzzy," that is that there is no black-and-white grammar to be described. The same is true, only more so, of languages as they are spoken by communities, where questions of individual and dialectal variability arise. But I assume for this paper that there is some definite body of knowledge to be described, even if statements of variability must form a part of the description.

which one of the many parsing engines should we use as *the* standard? SIL's AMPLE has the longest history, however it is incapable of describing real phonological rules. The Xerox finite state transducer, XFST, was developed in the late 1990s as proprietary software, and can describe phonological rules.[5] Another finite state transducer is the Stuttgart SFST program [24].[6]

The problem is that while these programs are useful, and certainly capable of creating testable grammars, none represents a real standard.

Moreover, none of these programs' notations looks to a linguist quite like a linguistic notation. Linguists are used to thinking in terms of phonological representations, parts of speech, morphosyntactic feature systems, declension or conjugation classes, allomorphs and phonological rules, and perhaps exception features. While all of these constructs *can* be represented in most modern morphological parsing engines, the *appropriate* representation is not always clear. For example, how should allomorphs and their conditioning environments be represented? The answer is clear in AMPLE (indeed, this is the only way in AMPLE to represent allomorphy), but it is not at all clear for the finite state transducers.

This brings us to my second criterion for morphological description:

Criterion 2. *There must be obvious formalisms which make it easy to handle the phenomena required for linguistic analysis.*

Again, the question arises as to what kind of formalism should be provided. It is a slight exaggeration to say that for nearly any mechanism which has been used in languages descriptions, there are proposals from theoretical linguists to do away with that mechanism. Phonological rules, for example, have been disposed of in Optimality Theory approaches to phonology. So what *is* the appropriate linguistic theory that a morphological parsing framework should implement?

It is safe to say that there is no consensus among linguists as to the One True Theory of morphology or phonology. This is in part an indication of our ignorance; we don't know enough yet to choose among the possibilities, and indeed the correct theory may not have appeared yet. But it is also the case that the term "correct" is part of the problem. In fact, linguists have explored several possible meanings of this term, using the terms *Observational Adequacy*, *Descriptive Adequacy*, and *Explanatory Adequacy* [5]. A description of a language can be considered observationally adequate if it generates all and only the sentences of the language—or, if one's interests are confined to morphology and phonology, then it is capable of generating all and only the possible inflected word forms of the language.[7] A theory of linguistics would meet this standard if it allowed observationally adequate descriptions of the grammars (or of the morphology and phonology) of all languages.

[5] A free implementation with most of the functionality of XFST exists as the Foma program, see `https://code.google.com/p/foma/`.

[6] `http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html`

[7] As compilers and users of corpora know, it is not clear that the set of all possible word forms of some language is a well-defined notion. Proper names, loan words in the process of being assimilated, and other boundary cases make this a dubious concept. For our purposes, however, I will assume that it is at least an approximation. Alternatively, one may conceive of a grammar which generates all and only the inflected forms of some static dictionary of words.

Given that for most languages, the number of inflected word forms is finite and (relatively) small, it would in principle be possible to create observationally adequate grammars by simply listing all the word forms. That is almost certainly not what humans do, at least not for languages which have any degree of inflectional or derivational morphology. The next step in Chomsky's hierarchy of adequacy is descriptive adequacy. The description of a language meets this standard if its analysis accounts for the intuitions that an adult native speaker has about the language; for example, that a *writer* is someone who *writes*, whereas a *grammar* does not mean someone who **gramms*. Determining those intuitions can be difficult; for example, it is not easy to know which inflected forms are memorized and which are derived by rule (or in some other way). Analogously, a descriptively adequate theory would allow descriptively adequate grammars to be written of all natural languages.

Finally, an explanatorily adequate *theory* would allow the selection from among candidate grammars of the correct descriptively adequate grammar.

As practiced by generative linguists, the science of linguistics is the search for an explanatorily adequate theory, based on descriptively adequate grammars. Until such a theory exists, and indeed until linguists have determined what a possible descriptively adequate grammar is, it would be inadvisable—not to mention impossible—to create a computational model for morphology and phonology which allowed for only descriptively adequate grammars. We simply don't know the range of possible variation, so trying to limit the range of what can be modeled could prevent the description of phenomena in some language. In other words, it is better to err on the side of excess descriptive power (potentially allowing the modeling of grammars which do not represent possible human languages) than on the side of insufficient power (thereby preventing the modeling of some languages).

An additional motivation for not attempting to attain the level of descriptive adequacy is that any such attempt would face a choice among several targets, all of which are moving. Not only are there multiple theories from which we could choose, but today's theories of morphology and phonology are not the same as those of ten or twenty years ago—and there is little reason to think that any theory in existence today will survive unaltered over the next ten or twenty years. Choosing one putatively descriptively adequate theory would therefore be very unlikely to result in a model which would stand the test of time.

In view of these facts, we have elected to create a model which gives observational adequacy only. The implications of limiting the scope in this way will become more apparent later, when I discuss the modeling of morphological reduplication; for now, I will state this point as the following criterion:

Criterion 3. *The model should allow for observational adequacy, not (necessarily) for the level of descriptive adequacy, even if this enables the description of grammars which may not correspond to any human language.*

Even if we did know what constituted a descriptively adequate grammar, our knowledge of the structure of a particular language at some point in time, while sufficient to describe that language's morphology to some satisfying level of detail, might be insufficient to create a descriptively or even observationally adequate grammar of the language. This is commonplace among field linguists, who wish to describe a language

based on limited field data; but it may also be true of comparatively well-known languages, where crucial data may be missing due to defects in previous descriptions, data which is unavailable, insufficient corpora, etc. An example of this is the notion of defective paradigms. It is known that for some languages, one form or another in the paradigm of particular verbs may not exist for various reasons [8, 23]. But it is nearly impossible to distinguish this situation in a corpus (particularly in a corpus of the size most field linguists will have) from the situation where a form is accidentally missing from the corpus.

While missing (or incorrectly assuming) paradigm gaps may be a relatively minor problem, there are many other situations where it will not be clear which analysis captures the facts of a native speaker's grammar better, particularly in less documented languages. Moreover, there may be disagreement among linguists as to the correct description; for instance, whether semi-regular forms are to be captured by special allomorphs of the stem or affix, or by rules governed by exception features (see section 3.2), or even whether such forms are memorized as entire words: Halle and Mohanan [9] argue for a rule-based analysis of English past tense verbs like *wept, kept,* and *slept*, while most linguists would be happy to treat those as being listed in the mental lexicon as irregular forms.

The point here, however, is not that linguists may disagree, or that they may have insufficient evidence to decide particular cases; rather, a morphological parsing system intended for use by real linguists should not force them into an analysis that they are not comfortable with, or which the evidence does not support. If it does, it will not be used. This point may be summarized by the following criterion:

Criterion 4. *If there is disagreement among linguists about the correct analysis of a particular language, the model should, where possible, allow for alternative analyses.*

The need for allowing the modeling of alternative analyses also may arise in the course of grammar development. It is especially unclear at the beginning of analysis of a previously undocumented or under-documented language what the phonological rules are by which allomorphs are derived. Usually only after observing a number of allomorphs conditioned by similar phonological environments does the linguist realize that a generalization can be made, allowing the allomorphs of distinct morphemes to be derived by a single set of rules.

This brings us to another point about modeling grammars, having to do more with the development of grammars than with the model itself: the need for testing, visualization and debugging methods. As with software development, grammar development is a process. Generalizations which seemed clear initially may turn out to be incorrect, while some correct generalizations may not become clear until much later. Finding out where a derivation is going wrong requires the ability to test a sequence of rules individually.[8] I summarize this point as follows:

[8] I assume here a model in which phonological rules are applied in a linear sequence, or possibly cyclically. But the same point—the need to visualize the application of grammar components, and to tease out the interaction among such components—applies to other models as well, e.g., to Optimality Theory models.

Criterion 5. *A grammar development environment must allow linguists to easily see the effects of individual grammar rules or other components of the grammar.*

This same point is also true for the reader of the grammar: it should be possible to visualize the derivation of individual forms. Moreover, it will be helpful to the reader (and also to the writer) if the components of the grammar can be displayed in some familiar form. While XML is a suitable format for archiving and transmission of grammars and other structured data (addressing Bird and Simons' concern for preservation, in particular that the need for longevity of the data), it does not lend itself to easy editing or even comprehension. Anyone who has looked at an XML-tagged textual document, and at the same document as a formatted PDF, will surely agree. This suggests two criteria; the first refers to static views of the grammar, the second to the ability to edit the grammar:

Criterion 6. *A grammar must be visualizable in a format which is familiar to linguists.*

Criterion 7. *A grammar development environment must allow easy editing of components of the grammar, in some format which is likely to be familiar to a linguist.*

There are several final criteria which are of a more practical nature. The first of these is connected with the fact that linguist must sometimes deal with written forms which do not represent all the phonemic contrasts of the language, or—worse—may even omit certain phonemes entirely. Languages written in Arabic scripts, for example, frequently omit short vowels. In many Brahmi-based scripts, consonants are assumed to be followed by a default vowel, unless there is an overt vowel letter following, or there is a mark indicating that no vowel follows. In practice, however, the mark representing no vowel is frequently omitted, which makes for ambiguity in the text representation. In sum, orthographies are seldom ideal, but they are what corpora are written in, and a linguist looking to create a practical morphological parser must deal with them. At the same time, the linguist may wish to have a more principled analysis, one which corresponds more closely to the phonological facts of the spoken language. While one could simply write two formal grammars, one for the orthographic form and one for the linguist's phonemic form, the two are likely to be quite similar. Certainly morphosyntactic features of affixes will be identical, as will many other features. A practical grammar development system should therefore allow parts of the description to be tagged for particular writing scripts; parts of the description which are not so tagged will be assumed to apply to all scripts. I capture this requirement as follows:

Criterion 8. *A grammar formalism must allow a single grammar description to be used with multiple scripts, with those parts of the description which apply only to a particular script being tagged for that script.*

Similarly, languages may have dialects which share most features, but differ in small ways. Again, it would be undesirable to have to write separate grammars for each dialect; rather, it should be possible to tag the differences:

Criterion 9. *A grammar formalism must allow a single grammar description to be used with multiple dialects, with those parts of the description which apply only to a particular dialect being tagged for that dialect.*

Finally, it is important in language documentation that the grammar description not suffer obsolescence. While the use of open-source software as a parsing engine partially alleviates this problem, since one can presumably re-compile the source code in the future, it is not a complete answer. First, there is no guarantee that the source code of such a parsing engine *will* re-compile; programming languages change, and so do libraries that the parsing engine may require.[9] In principle, such problems could be overcome by running old versions of all the software; in practice, this solution is too complex for use.

A second issue with the use of open-source software for preventing obsolesence is that even this may not suffice for the long term. What will the computing landscape be in a hundred years, or five hundred? Imagine if we had to reconstruct programs which were written for Babbage's mechanical computers. It is safer to assume much less about what facilities will be available; and text (Unicode) data in plain text markup formats is far safer than untagged data in the programming langauge of a present-day parsing engine (cf. Bird and Simons' point about "Markup," specifically their calls for the use of plain text markup, as well as their point about "Openness," their term for the avoidance of proprietary formats). I represent this criterion as follows:

Criterion 10. *A plain text with markup representation of the grammar is to be preferred to a representation in the programming language of some particular software.*

While one could no doubt add criteria for grammatical modeling and the development of grammatical analyses, the above list will suffice for now. I will now describe a methodology which we have developed and are using, and how it satisfies these criteria.

3 Satisfying the Criteria for Grammatical Descriptions

At the University of Maryland, we have developed a technology which allows the statement of language-specific aspects of morphological and phonological descriptions in a transparent, parser-independent and linguistically motivated formalism. This technology has allowed us to satisfy many of the criteria outlined in the previous section. Planned extensions will further increase the ability to satisfy these criteria, but the existing system is robust enough to have been used for constructing descriptions from which (along with XML-based dictionaries) parsers for five languages have been built and tested automatically.

In broad overview, descriptions are written in an XML-based formal grammar format and validated against an XML schema. These descriptions are then read into a converter program, along with lexicons. The converter functions in a way analogous to a modern programming language compiler: it converts the XML-based description into a corresponding internal representation, which is in turn output in the form required by

[9] The author experienced this with a parser he wrote in the 1990s. The parser and its user interface was written in three programming languages: C, Prolog, and Smalltalk. Within one year, all three languages changed in ways which broke the parser. While changing the parts written in C would not have been difficult (it involved a change from 16 bit integers to 32 bit integers), the changes in Prolog (having to do with calling the C code from Prolog) were extensive, and the Smalltalk vendor went out of business, leaving only another vendor's incompatible version.

an external parsing engine (currently the Stuttgart Finite State Tools, SFST). Running the parsing engine's "compiler" over this output results in a form usable by the parsing engine for morphological analysis.

For three of the five languages that we have worked on, we have additionally employed Literate Programming [17]: we embed the formal grammar as XML fragments into an XML-based (DocBook, [25]) descriptive grammar. Each fragment appears in the text of the overall document next to the description of the grammar construction that the fragment instantiates, allowing the descriptive grammar to explain the formal grammar, while at the same time allowing the formal grammar to disambiguate the descriptive grammar where necessary. The fragments appear in an order which is useful for expository purposes; e.g., fragments having to do with nominal affixes appear in the nouns chapter, while fragments containing verbal suffixes appear in the verbs chapter. The fragments can be extracted by an XSLT transformation and placed in the correct order for computational processing in a file to be read by the converter program mentioned above.

In addition, the interlinear and in-line examples found in our descriptive grammars are extracted and used for parser testing. (Additional testing is done by running the parser over corpora.)

The use of Literate Programming and the extraction of examples from the descriptive grammar is discussed elsewhere [6, 19, 21]. The remainder of this document discusses the formal grammar and converter implementation, plus planned enhancements.

3.1 Formal Grammar Implementation

As mentioned above, the formal grammar of a language is a linguistically-based description of the morphology and phonology of that language, written in XML and validated against an XML schema. After briefly describing this schema, I show how this approach accomplishes most of the design goals outlined in the earlier sections of this paper.

The XML schema, to be documented in [20], organizes information about the grammar into five general categories:

1. Morphosyntactic Feature System
2. Grammatical Data
3. Morphological Data
4. Phonological Data
5. Lexical Data

The Morphosyntactic Feature System specification is slightly simplified from the model given in [4, 13, 15], and defines the possible morphosyntactic features including both simple features (e.g., binary features) and feature structures (features whose values consist of other features).

The Grammatical Data module supplies information about parts of speech (typically just those that accept affixes). This information includes which of the morphosyntactic features defined in the Feature System are possible for each part of speech. It also points

to the affixes that each part of speech takes (these are defined in the Morphological Data module), and specifies their morphotactics.

The Morphological Data module defines the derivational and inflectional affixes of the language. They are defined here, rather than in the Grammatical Data module, so that they can be shared across parts of speech. For example, in Tzeltal (an ergative language of Mexico), transitive and intransitive verbs share absolutive agreement suffixes. If these two verb classes are defined as different parts of speech in the grammatical module, the shared absolutive suffixes can be defined once in the Morphological Data module, and used for both parts of speech.[10]

As will be discussed in greater detail below, the model allows for "ordinary" prefixes, suffixes and infixes; these may be defined either as underlying forms, with any allomorphs derived by phonological rules, or as allomorphs which appear in particular phonological environments. In addition, there is allowance for affixes defined as processes, that is, as morphological rules which may attach constant phonological material (as with ordinary affixes), but may also copy or delete parts of the base, and which can therefore model processes such as reduplication.[11] Allowing affixes to be represented as either a set of allomorphs, or as underlying representations with allomorphs derived by phonological rule, and allowing affixes to also be represented as processes, are two examples of the way the model allows for multiple analyses of a language's grammar (criterion 4).

In addition, the Morphological Data module allows for the definition of inflectional classes (declension and conjugation classes); since they are defined here, they can be shared by multiple parts of speech, as in Pashto, where nouns and adjectives have more or less similar declension classes. Finally, any "stem names" are defined in the Morphological Data module. These allow the implementation of irregular stems for certain lexemes, such as the diphthongized forms of Spanish verbs.[12]

The Phonological Data module defines the phonemes and/or graphemes of the language, boundary markers (used to delimit morpheme boundaries), and phonological rules. The latter come in three varieties: rules which change input phonemes (or graphemes) to other phonemes (graphemes); rules which epenthesize phonemes (or graphemes); and rules which delete phonemes (graphemes).

Notice that the model does *not* define phonological features. While the model could be extended to allow this (as well as the definition of phonological rules using such features), this omission is intentional. First, the nature of phonological features is still in doubt; it is not clear whether they are hierarchically structured, for example. Second, the use of phonological features in parsing would preclude the use of most present-day

[10] The ergative agreement prefixes on Tzeltal transitive verbs are homophonous with the possessive prefixes on nouns. Depending on how their morphosyntactic features are defined, it would also be possible to define these prefixes once in the Morphological Data module, and use them for both nouns and transitive verbs.

[11] This is discussed further below; however, this has not yet been implemented in the converter program.

[12] An alternative analysis would derive some or all irregular stems by phonological rules, probably conditioned on lexeme-specific rule exception features; this can also be modeled.

parsing engines. [13] In generative phonology, the principle use of phonological features is to define natural classes of phonemes, which are then used in the inputs and environments of phonological rules. Such a definition may be termed intensional. The approach taken in our model may be described as extensional: natural classes are instead defined by listing their member phonemes (or perhaps graphemes).

It is sometimes convenient to define natural classes, contexts (regular expressions over phonemes, graphemes and natural classes), and environments (the combined left and right context of some phonological process) once, and re-use these definitions for multiple rules or allomorph environments. These elements can therefore be defined in the Phonological Data module, and referred to where used (by their XML ID); but they may also be simply written out in rules or allomorph environments, which is convenient when such an element is needed only once.

As discussed above, the linguist must sometimes deal with scripts which differ in their ability to represent the phonology (criterion 8). The phonology module is frequently the locus of such differences; such differences are handled by tagging affected elements for the script for which they are relevant. Script-specific elements are therefore tagged with a 'script' attribute; they can then be included, or not, by removing or retaining them during a pre-processing stage, prior to their being read by the converter program. Dialect-specific elements are handled in the same way (cf. criterion 9).

The Lexical Data specification is derived from the ISO Lexical Markup Framework standard [14], supplying just the information about suppletive word forms and stem allomorphs required for morphological parsing. Words which require no special treatment (i.e., "regular" words) could be loaded in this module, but they are usually handled more quickly by pre-processing a dictionary into whatever form is required by the parsing engine, and them loading them into the parser directly, during the parser compilation phase.

Affixes as Processes. As discussed earlier, the goal of this framework is to attain the level of observational adequacy (cf. criterion 3). This is perhaps nowhere more apparent than in the treatment of affixes as processes, particularly reduplication. Reduplicative morphology in real languages ranges from complete reduplication, where an entire word is pronounced twice (used for a sort of pluralization in Bahasa Indonesian), to forms in which a single phoneme of the base is copied, perhaps augmented by some constant phoneme or sequence of phonemes (as was found in the perfect of some Ancient Greek verbs). Complications abound; for example, it is not unheard of for both the reduplicant and its correspondent in the base (the input to the reduplication process) to undergo some phonological process for which only one or the other is in the appropriate phonological environment.

Among theoretical linguists, it has become a cottage industry to develop theories which limit the possible forms of reduplication to all and only forms which are attested in languages of the world. In contrast, the model described here makes no attempt at limiting the power of reduplication; the formalism is sufficiently powerful allows almost anything to happen, even for the phonemes of a (fixed length) word to be reversed,

[13] One exception is the Hermit Crab parser, described in `http://www-01.sil.org/computing/hermitcrab/` . This parser was originally developed by the author in the 1990s, but it has been re-implemented more recently in SIL's FLEx tool.

something which has never been observed in real languages. That is, the formalism is observationally adequate, but probably not descriptively adequate.

The formalism is based on [18]. It involves matching an input word (the base) with a regular expression over phonemes, graphemes and natural classes. The output is formed by concatenating copies (possibly altered) of the parts of the base which matched the regular expression in some pre-defined order, possibly combined with other strings or phonemes.

An example may help; for ease of exposition, I will use a notational formalism, rather than the XML formalism.

Suppose we have a rule of reduplication which copies the first consonant (if any) of the base, adds the vowel 'e', and appends this to the base. Conceptually, we may capture this with the input regular expression '(C) X', where 'C' is assumed to have been defined as the natural class of consonants, 'X' is a variable matching any sequence, and the parentheses around the 'C' represent optionality. The content parts are implicitly numbered; the 'C' as part 1, the 'X' as part 2. The output may then be specified as '1 e 1 2'. Note that if the base is vowel-initial, the optionality of the '(C)' in the regular expression means that part 1 would be a null match, giving what is presumably the desired result.[14] Had the input regular expression been 'C X' (with the consonant obligatory), the rule would not match a vowel-initial base, meaning that this affix process would not apply to such a base.

In addition to copying part of the base to the output or adding specific strings (represented as phonemes, graphemes, and boundary markers), process affixes allow modification of input parts which are copied over. Suppose for example the grammar defines phonemes /p/, /p^h/, /t/, /t^h/, /k/ and /k^h/, and suppose further that the output of the above process had been defined conceptually as

`[1 (/`p^h`/→/p/,/`t^h`/→/t/,/`k^h`/→/k/)] e 1 2`

where the square brackets are used here for grouping the phonological process of deaspiration with the output part to which the process applies. Applied to a base beginning with an aspirated consonant such as /phu/, this would give the reduplicated form /pephu/; applied to a base which began with an unaspirated consonant such as /grap/, the result would be /gegrap/.[15]

The use of this process affix formalism is not limited to rules of reduplication; it can also be used to describe the situation where an affix simply modifies its input, without copying or adding additional phonemes. For example, the following rule describes an

[14] The parsing of the base into the parts which correspond with the regular expression must prioritize contentful parts of the regular expression (like the part in the example which matches a consonant) over less contentful parts (like the variable matching any string), lest the output be ambiguous. One can imagine regular expressions which would remain ambiguous even under such prioritization, such as '(C)(C)X' matched against a base beginning with a single consonant followed by a vowel; presumably what would be intended in such a case would be '((C)C)X' or '(C(C))X'. Some error checking will therefore be required, to avoid such ambiguous regular expressions.

[15] This is known to linguists as Grassmann's Law, and the examples given are from Ancient Greek (ignoring vowel length for purposes of exposition).

affix formed by palatalizing the stem-final consonant, under the assumption that the phoneme inventory has been defined as including /p/, /p^{y}/, /t/, /t^{y}/, /k/ and /k^{y}/:[16]

```
X C
1 2 →
1 [2 (/p/→/p^y/,/t/→/t^y/,/k/→/k^y/)]
```

We are now in a position to understand why this system achieves only the level of observational adequacy, not (probably) descriptive adequacy. Consider the following description of a putative process affix process (‘C’ and ‘V’ are assumed to have been defined as the natural classes of consonants and vowels respectively):

```
C V C C V C
1 2 3 4 5 6 →
6 5 4 3 2 1
```

This rule takes a six phoneme input and reverses it. It is highly unlikely that such a process exists in any human language, but it can be easily described in the notation used here (or in its XML equivalent). Most generative linguists would prefer a theory which disallowed (or at least made highly unlikely) statements of such nonexistent processes. The problem is that we don’t have such a theory, but we still wish to be capable of writing grammars—which is wy we have settled for the level of observational adequacy; that is, we are content to describe *all* languages, but make no attempt to limit possible descriptions to *only* natural languages.[17]

3.2 Converter Implementation

This section describes how the converter takes as input a formal grammar stated in XML, and outputs the grammar in the form required by a parsing engine.

There are several reasons for using a formalism which requires a converter in order to be usable by a parsing engine, rather than a formalism which is directly interpretable by the parsing engine. First, criterion 2 in section 2.2 dictates that the formalism needed to handle linguistic structures should be (relatively) obvious. By constructing our linguistically based formalism in XML, we hope that the formal grammar mechanism will be more easily learned by most working linguists, and grammars written in that formalism will be more easily understood by linguists.

Another reason for using a formalism such as the one described here, rather than the programming language of some parsing engine, is to prevent the formal grammar from becoming obsolete when the parsing engine becomes obsolete, as it (like any software) inevitably will. This is another of the criteria given above (10), as well as helping answer Bird and Simons’ points about “Markup” and “Openness.”

[16] This rule is based on Oaxacan Mixe, as described by Dieterman [7, p. 39].

[17] It is at least possible that the explanation for the non-existence of processes reversing their input is due to factors other than the human language capability, e.g., the fact that such systems have no plausible diachronic source. The fact that certain reversals do occur in language games, e.g., the reversal of two consonants across a vowel, might be taken as such evidence. Thus, the search for a formalism which prevents such unattested processes might be misguided.

As discussed above, we are currently using the Stuttgart Finite State tools (SFST) as our parsing engine. Given that a formal grammar written in XML cannot be directly interpreted by SFST, there is a need for converting the XML representation into the representation required by SFST. In principle, this could be done using Extensible Stylesheet Language Transformations (XSLT). In practice, it has been easier to do the transformation in Python. The converter is written as an object-oriented program, where the classes correspond one-for-one to the elements defined in the XML schema. The conversion takes place in two phases. In the first phase (corresponding roughly to the "front end" of a modern programming language compiler), the XML representation is converted into the internal representation as Python objects. References from some objects to definitions made elsewhere (e.g., from natural classes to the phonemes they are composed of) are converted into pointers to the corresponding definitions (analogous to "object binding" in modern compilers). Most errors and warnings are issued at this stage.

In the second phase, corresponding to the "back end" in a modern compiler, the converter writes the parsing engine's code to output. Some optimization is done at this point, in the sense that the output code is optimized for "compilation" by the parsing engine.[18]

Since the first phase maps between two fairly congruent representations, it makes use of a mostly declarative format for the individual classes; most of the non-declarative code for converting from XML to the internal format is contained in an abstract superclass.

The second phase, however, can be more complex, since it maps between two representations which at times diverge strongly. Where these representations are similar, the conversion is fairly straightforward. Consider for example the following code:

```
def SFSTOutput(self, sFormat, ExtraArg=None):
   """
   Output this context in the form expected by SFST, i.e.,
   ( X | Y | Z )
   """
   if sFormat == 'AsRegex':
      self.SFSTOutputList("PhonologicalContexts",
                          "(",
                          "|",
                          ")",
                          sFormat)
   else:
      AbstractClasses.LangClass.SFSTOutput(sFormat, ExtraArg)
```

This `SFSTOutput()` function is defined for the class `AlternativeContexts`, which encodes a set of alternative phonological contexts forming part of the

[18] There is no attempt to ensure that the final transducer as compiled by the parsing engine will be optimal, e.g., by tweaking the alignment of lexical and surface sides of lexical items. This might have significant effects if the citation form of lexemes includes a prefix, which is removed to form the stem.

environment of a phonological rule (or a phonologically determined allomorph); for example, the context of a long vowel or a vowel plus consonant. The function is called with an argument list specifying a format (and an optional extra argument). The only format this particular function knows about is called `'AsRegex'`; any other format is referred by the `'else'` clause to the superclass of `AlternativeContexts`, here `AbstractClasses.LangClass`. For this `'AsRegex'` format, the function needs to output the alternatives in the format which SFST expects for a regular expression, namely a parenthesized list with list members separated by the character '|'. Since outputting of lists with various delimiters is a common task in the converter, the details of outputting the list (such as the need to output the separator character after every member of the list except the last) is here delegated to a more generic function, `SFSTOutputList()`, which takes as additional arguments the character which starts the list (here an open parenthesis), the separator character ('|'), and the character which marks the end of the list (a close parenthesis).

The XML elements which constitute the alternatives (represented by `X`, `Y` and `Z` in the quoted comment) will be recursively output by `SFSTOutput()` functions defined on whatever classes these individual contexts belong to. This is the general pattern for how the `SFSTOutput()` function is written on all classes: there may be several cases, depending on the purpose for which the element is being output (although here there is only one case, the `'AsRegex'` case). Within each such case, the class specifies some of the output (here, the open and close parentheses, and the pipe symbol '|'), while the output of elements which may be contained by an element of the specified class are delegated to those classes (here, the classes of the embedded contexts).

A more complex conversion is needed for other constructs. The code for converting Affix Allomorphs, for example, has four cases. One of these cases constrains the allomorph to appear in its required environment. This requires outputting the allomorph itself, as well as calling the environment class to output the phonological environment, in essence creating a rule which blocks the allomorph if this environment is not satisfied. The mechanism for accomplishing this is that all allomorphs are initially inserted in the transducer bracketed by marks which would block a derivation containing them from appearing at the end of the derivation. The marks are erased for allomorphs whose environment is satisfied; finally, any words still containing marks are removed from the network. This is precisely the sort of non-obvious solution that is one of our motivations for the use of a linguistically informed formalism, which must be automatically converted into the parsing engine's formalism.

Another example of a non-obvious solution concerns rule exception features. These are lexical features (that is, features assigned to particular roots or stems in the lexicon) which either trigger the application of particular rules (positive exception features) or prevent the application of certain rules (negative exception features). Consider for example diphthongization in Spanish verb paradigms. For a certain set of verbs (those which had a long stem vowel in Latin), the vowel /e/ diphthongizes to /ye/ (spelled 'ie') when stressed, while the vowel /o/ diphthongizes to /we/ (spelled 'ue') when stressed.[19] There is no phonological indication of which verbs undergo this rule and which do not;

[19] There are exceptions to this generalization, chiefly where an 'n' becomes an 'ng', for example: tiene "he/she has", tengo "I have", both with stress on the first syllable.

hence this information must be stored in the lexicon, either in the form of listed allomorphs, or—for a rule-governed analysis—in the form of positive exception features. Thus, contar∼ cuento "to count/ I count" (a diphthongizing verb) vs. montar∼ monto "to mount/ I mount" (a non-diphthongizing verb).

Linguists often conceive of such exception features as being part of the phonological material, and therefore visible to the phonological rules which may require them. The obvious solution would then be to assign such features to the "surface" side of the transducer representing the word of the language, and to apply phonological rules on this side so that rules which need to refer to exception features can "see" them. However, exception features must be invisible to phonological rules which do not require them, since the features might otherwise appear to be phonemes, and such rules would therefore not match the lexical entries containing these phoneme-like exception features.

The problem is that phonologists really conceive of words not as strings composed of sequences of phonemes and exception features, but rather as sequences of phonological features representing such properties as voicing, nasalization, and place and manner of articulation, with each such feature on a different plane;[20] exception features are on yet other planes. The voice features of two adjacent phonemes are therefore adjacent on the voice plane, regardless of any exception features, so that the problem that adjacency between phonological features would be blocked by exception features or other phonological features is avoided.

But practical finite state transducers, such as XFST and SFST, have only two planes (or levels): a lexical side and a surface side. In such a model, exception features on the surface side would block adjacency between phonological features as seen by phonological rules composed on that side (indeed, distinct phonological features would get in each other's way, if they were to be represented in such transducers).

There are several ways that this issue of adjacency could be treated in a transducer with only two levels. One way would be to construct phonological rules such that exception features (and other grammatical information relevant to rule application) are allowed to intervene between any two phonemes in the regular expressions representing the rule environments. While tools such as SFST allow rules to be constructed in this way, in practice it tends to make compilation very inefficient. Our converter therefore handles exception features in a different way, which may be less obvious at first sight. When lexical entries are imported from a machine-readable dictionary, the conversion process constrains any exception features to appear on the lexical side of lexical entries.[21] At the beginning of the derivation, the converter collects these lexical entries into a lexical transducer *L*, to which the phonological rules are applied in sequence by composing each rule on the surface side of *L*. When translating a rule which is sensitive to an exception feature, the converter first outputs SFST code which composes a filter on the lexical (underlying) side of *L*, thereby selecting a subset *L1*. This filter is a regular expression accepting all paths through the lexical transducer which contain the relevant exception feature. The converter also outputs SFST code to create the complement of

[20] I abstract away here from questions of the typology of features, which are generally held to have still more structure than what is described here.

[21] A given lexical entry may have several such exception features; see for instance Harris's [11] analysis of Spanish verbal morphology.

this subset—call this *L2*—by subtracting *L1* from *L*.[22] The sensitive rule is then applied to *L1* by composing the rule on the surface side of *L1*. Finally, *L1* and *L2* are unioned to form a new lexical transducer *L*, to which the remaining phonological rules will be applied. At the end of the derivation, when the exception features have done their work, they are removed from the underlying side of *L*.

Linguists also occasionally find the need to write phonological rules which are sensitive to particular parts of speech, or to certain morphosyntactic features. Allowing this sensitivity can be done in a way analogous to that used for rule exception features: splitting the lexicon into two halves allows for rules which do not display such sensitivity (usually the vast majority of such rules) to implicitly ignore the part of speech or morphosyntactic features.

3.3 Further Work

The description of the work we have done thus far leaves several of the criteria for a morphological and phonological description system unsatisfied. In particular, the criterion that there be a debuggging environment (5); that the formal grammar be in a form that is easily visualized by linguists (cf. Bird and Simons' point about rendering linguistic documents in human-readable form, and my criterion 6 that the grammar be visualizable); and the criterion calling for a grammar editing environment (7), have not been addressed, unless one considers viewing and editing XML to be something the average linguist will enjoy, and that editing the SFST code is a suitable means of debugging.

In addition, while the XML schema supports all the elements described above, not all such elements are supported by the converter as yet. In particular, the support for affixes as processes is missing, and the support of listed stem allomorphs is not complete. We plan to address these shortcomings in future work; I outline the plans here.

First, conversion support for the remaining elements needs to be added.

Secondly, when a descriptive grammar (which we write using a slightly modified version of the DocBook XML schema) is converted to PDF presently, the formal grammar is output in its native XML form. Needless to say, linguists have a hard time interpreting this. For instance, rather than outputting a phonological rule as some complex XML structure, most linguists would prefer to see it in something like this format:

z → s / VoicelessC ___

We produce the descriptive grammars by converting the XML source into LaTeX (or more precisely, XeLaTeX), and then producing a PDF from that. The conversion from XML to LaTeX format is done by XSLT transformations, using the `dblatex` program.[23] We therefore need to add XSLT transformations to convert our formal grammars into LaTeX format; alternatively, we could process them using another program (such as our existing Python-based converter).

A grammar development environment which knows what elements are possible at any point would also be an improvement over editing the XML formal grammar in a programmer's editor. Displaying the elements of the formal grammar in something like

[22] Alternatively, by composing the converse of the filter on the underlying side of *L*.

[23] The dblatex program is open source; see `http://dblatex.sourceforge.net`

the format a linguist expects (probably an approximation of the planned PDF format) would also help make grammar editing accessible to more linguists. We currently edit the DocBook descriptive grammars in the `XMLMind` editor.[24] This program uses Cascading Style Sheets (`css`) to display DocBook structures in a semi-wysiwyg editable fashion, and XML schemas (in the Relax NG, or RNG, format), to determine what elements can be added at any place in the structure. We already have an RNG schema for our formal grammar, so the remaining work would be to specify CSS styles for the elements.

Thirdly, we intend to build a grammar debugging environment, which will allow linguists to generate and view paradigms, and help determine why expected forms are not being produced. This will involve automatically compiling subsets of the grammar—e.g., compiling the grammar for a single part of speech, using a single lexical item for the sake of speed. To show the steps in a derivation, the debugging system would compile the transducer multiple times, with one additional phonological rule applied each time.

Finally, I have not said anything about Bird and Simon's call for linguistic descriptions to support the need for terminology to be defined. To some extent, this can be done in the descriptive grammars associated with our formal XML-based grammars. However, we also plan to add a simple enhancement to our current XML schema for tagging appropriate elements, such as morphosyntactic features, by linking to their definition, e.g., in the ISOcat data category registry of linguistic terminology (`http://www.isocat.org/`).

4 Conclusion

I have laid out a number of design critera for a morphological and phonological system to be used in language documentation, and shown how the system our team has developed satisfies most of those criteria. I have also described how we plan to further develop this system to satisfy the remaining criteria.

A few aspects of the system are still in flux; specifically, the representation and conversion of lexically listed stem allomorphs, and the conversion of process affixation into the form needed by a parsing engine. Satisfying the remaining design criteria—for example, by providing a debugging system—would make the system still more usable. As the system becomes stable and more usable, we expect to make it freely available through an open source license (which one is yet to be determined).

References

1. Beesley, K.R., Karttunen, L.: Finite State Morphology. University of Chicago Press, Chicago (2003)
2. Bird, S., Simons, G.: Seven dimensions of portability for language documentation and description. Language 79(3), 557–582 (2003)
3. Blevins, J.: A reconsideration of Yokuts vowels. International Journal of American Linguistics 70(1), 33–51 (2004)

[24] This is a commercial program; see `http://www.xmlmind.com/xmleditor`. A similar program is the oXygen XML Editor, see `www.oxygenxml.com`.

4. Burnard, L., Bauman, S.: TEI P5: Guidelines for electronic text encoding and interchange (2013)
5. Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge (1965)
6. David, A., Maxwell, M.: Joint grammar development by linguists and computer scientists. In: IJCNLP, pp. 27–34. The Association for Computer Linguistics (2008)
7. Dieterman, J.I.: Secondary palatalization in Isthmus Mixe: a phonetic and phonological account. SIL International, Dallas (2008), http://www.sil.org/silepubs/Pubs/50951/50951_DietermanJ_Mixe_Palatalization.pdf
8. Halle, M.: Prolegomena to a theory of word formation. Linguistic Inquiry 4, 3–16 (1973)
9. Halle, M., Mohanan, K.P.: Segmental phonology of modern english. Linguistic Inquiry 16(1), 57–116 (1985)
10. Hankamer, J.: Finite state morphology and left to right phonology. In: Proceedings of the Fifth West Coast Conference on Formal Linguistics. pp. 29–34 (1986)
11. Harris, J.W.: Two theories of non-automatic morphophonological alternations. Language: Journal of the Linguistic Society of America 54, 41–60 (1978)
12. Harris, Z.: Yokuts structure and Newman's grammar. International Journal of American Linguistics 10, 196–211 (1944)
13. ISO TC37: Language resource management — Feature structures — Part 1: Feature structure representation (2006)
14. ISO TC37: Language resource management — Lexical markup framework, LMF (2008)
15. ISO TC37: Language resource management — Feature structures — Part 2: Feature system declaration (2011)
16. Karttunen, L.: The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. In: Kaplan, R.M., Butt, M., Dalrymple, M., King, T.H. (eds.) Intelligent Linguistic Architectures: Variations on Themes, pp. 287–300. CSLI Publications, Stanford (2006)
17. Knuth, D.E.: Literate Programming. Center for the Study of Language and Information, Stanford (1992)
18. Marantz, A.: Re reduplication. Linguistic Inquiry 13, 435–482 (1982)
19. Maxwell, M.: Electronic grammars and reproducible research. In: Nordoff, S., Poggeman, K.-L.G. (eds.) Electronic Grammaticography, pp. 207–235. University of Hawaii Press (2012)
20. Maxwell, M.: A Grammar Formalism for Computational Morphology (forthcoming)
21. Maxwell, M., David, A.: Interoperable grammars. In: Webster, J., Ide, N., Fang, A.C. (eds.) First International Conference on Global Interoperability for Language Resources (ICGL 2008), Hong Kong, pp. 155–162 (2008), http://hdl.handle.net/1903/11611
22. Newman, S.: The Yokuts Language of California. Viking Fund, New York (1944)
23. Rice, C., Blaho, S. (eds.): Modeling ungrammaticality in Optimality Theory. Advances in Optimality Theory. Equinox Press, London (2009)
24. Schmid, H.: A programming language for finite state transducers. In: Yli-Jyrä, A., Karttunen, L., Karhumäki, J. (eds.) FSMNLP 2005. LNCS (LNAI), vol. 4002, pp. 308–309. Springer, Heidelberg (2006)
25. Walsh, N.: DocBook 5: The Definitive Guide. O'Reilly, Sebastopol, California (2011), http://www.docbook.org/
26. Weber, D.J., Black, H.A., McConnel, S.R.: AMPLE: A Tool for Exploring Morphology. Summer Institute of Linguistics, Dallas (1988)
27. Weigel, W.F.: The interaction of theory and description: The yokuts canon. Talk Presented at the Annual Meeting of the Society for the Study of the Indigenous Languages of the Americas (2002)
28. Weigel, W.F.: Yowlumne in the Twentieth Century. Ph.D. thesis, University of California, Berkeley (2005)