Interoperable Grammars

Michael Maxwell
Center for Advanced Study of Language/
University of Maryland
College Park, Maryland, USA
mmaxwell@casl.umd.edu

Anne David
Center for Advanced Study of Language/
University of Maryland
College Park, Maryland, USA
adavid@casl.umd.edu

Abstract

For languages with significant inflectional morphology, development of a morphological parser is often a prerequisite to further computational linguistic capabilities. We focus on two difficulties for this development: the short lifetime of software such as parsing engines, and the difficulty of porting grammars to new parsing engines. We describe a methodology we have developed to promote portability, using a formal declarative grammar written in XML, which we supplement with a traditional descriptive grammar. The two grammars are combined into a single document using Literate Programming. The formal grammar is designed to be independent of a particular parsing engine's programming language, thus helping solve the software lifetime and portability problems.

1 Grammar Development

After decades of widespread effort in computational linguistics, it is clear that progress has been made in areas ranging from the building computational lexical resources, to applications such as machine translation. While all this is beneficial, it is not without drawbacks: some resources which were developed at great expense, and which proved useful in the past, are no longer usable. This is perhaps nowhere more true than with grammars. Numerous computationally implemented grammars have been written, often at great expense; but nearly all of these grammars are tied to particular parsing engines, and their usefulness ends with the obsolescence of that parsing engine.

Recently, many computational linguists have turned from hand-crafted, labor intensive grammars to grammars automatically induced from annotated corpora, particularly in syntax. When the grammar is learned from a corpus, the obsolescence of a parsing engine may be a lesser issue, because when someone invents a better parsing tool, a grammar for that new tool can be induced from the same annotated corpus. This changes the issue from the obsolescence of grammars to the obsolescence of annotated corpora, and progress has been made in this area.

Automatic grammar induction has been more popular for syntax than for morphology. This is not to say that there has been no research into the learning of morphology; see for example Creutz and Lagus 2007, Goldsmith 2001, Goldsmith and Hu 2004, and the papers in Maxwell 2002. But research on morphology learning has not had the same impact that syntax learning has had. Accordingly, most wide coverage morphological parsers for languages with significant amounts of inflectional morphology are probably still built by hand; and barring a breakthrough, this seems likely to continue, at least for the near future.

Thus, for languages with significant inflectional morphology, a morphological parser[1] is a prerequisite to serious natural language processing. And to the degree that a language has complex morphology, the grammars for these parsers are difficult and time-consuming to build. One would therefore like to preserve this investment.

Unfortunately, the development of computer-processable morphological grammars is often tied to the programming language of a particular morphological parser, or to a general purpose computer programming language, such as Prolog or Haskel (see e.g. the papers at http://www.cs.chalmers.se/~markus/FM/index.html). If the particular morphological parser (or transducer) never became obsolescent, or if there were a standard descriptive lan-

[1] One commonly builds a morphological transducer, that is, a program which functions to both parse and generate inflected words. However, because it is more familiar, in this paper we will use the term 'parser.'

guage that all parsers used, this might not be problematic. But neither of these conditions is true. In the past 25 years, there have been at least half a dozen mutually incompatible morphological parsing languages, ranging from SIL's AMPLE (Weber Black and McConnel 1988) and PC-KIMMO (Antworth 1990) to Xerox PARC's xfst (Beesley and Karttunen 2003) Nor have developers of morphological parsing engines agreed upon a common language; two recent entries, the Stuttgart Finite State Transducer (http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html) and the OpenFst Library (http://www.openfst.org) provide still different programming languages. Some changes are motivated by enhanced capabilities, but others seem to be more an issue of style.

Two problems arise out of the mutual incompatibility of programming languages for different parsing programs: an Interoperability Problem, and a Half-Life Problem. The Interoperability Problem refers to the fact that a grammar written for one parsing engine cannot be used in another parser without re-writing; and for now, at least, that re-writing must be done by hand, since there are no automatic interpreters between parsing engine languages. A grammar written for the Xerox transducer, for example, will not run on the Stuttgart transducer without considerable modification.

The Half-Life Problem arises from the fact that software (in particular, parsers) becomes obsolete. While we are not aware of formal investigations, we estimate the average lifetime for language-based computational tools at five or ten years. In part, this is due to the (lack of) longevity of the underlying software.[2]

Software obsolescence can be postponed by the judicious choice of programming languages, avoiding OS-specific commands, the use of Open Source software, and the use of OS emulators. However, this can only prolong the life of a program, not extend it indefinitely. Few if any programs that were written in 1980 (twenty-seven years ago) still run on today's computers.

One might argue that software half-life is unimportant, since twenty years from now it may be possible to generate a morphological parser automatically from a corpus. Perhaps, but this remains to be seen. Meanwhile, the time and effort that go into writing grammars mandates that the grammars be usable long after the project is completed.

Another motivation for building longevity into parsing tools is that they constitute a description of (part of) the grammar of a language, in two senses. First, the grammar the parser uses constitutes a formal description of the language's morphology or syntax. Second, it can be used to analyze language texts, and—if it supports a generation mode—to produce paradigms. That is, a parser is an active description, not just a static one. But in their seminal paper, Bird and Simons 2003 point out that language data in computer-readable form can become unusable much more quickly than printed descriptions. In contrast, scholars of today can understand grammars and corpora penned thousands of years ago. Thus, while a parser constitutes a description of a language, it is—at present—an ephemeral description.

There is little doubt that future parsing engines will be improvements upon today's parsers. We are not suggesting that we need to build parsing engines which will continue to be used decades from now. Rather, we are suggesting that the language-specific information that goes into a parser—the grammar—should be written in such a way that it can be easily ported to future parsing engines. From this perspective, the Half-life Problem is really the Interoperability Problem in a different guise: interoperability between grammars written today, and tools which are yet to be built.

In summary, the problem is that while computational grammars have real worth, each parsing engine uses a different programming language. One might therefore conclude that there is no hope of providing a generic programming language for grammars. We claim that this conclusion is wrong; it is time now to think how we can write such grammars that will not only be interoperable with today's parsers, but with future parsing engines; and we provide a first cut at what such a programming language for morphology could look like.

One reason for optimism is the fact that morphological parsing tools now incorporate most of the capabilities that linguists have found necessary for morphology and phonology (albeit clumsily in some cases, e.g. morphosyntactic features), and that a morphological grammar written in a generic

[2] The first author, (Maxwell) was involved in a project in which two of the programming languages became defunct before the program was even complete; the cost of porting to alternative dialects of the programming languages was deemed prohibitive.

way can therefore be compiled into the programming language of current morphological parsers. At first glance, it might seem that this claim is simply wrong, because linguists have yet to come to agreement on the correct theory; like software, linguistic theories have a short half-life! Since the mid-1950s, there have been several generations of theories about what the phonological 'atoms' are, how many levels of structure are important, and how representations are translated between those levels. Atomic phonemes of the 1950s were replaced by distinctive feature matrices, which were in turn superseded by autosegmental representations. Nevertheless, these changes were primarily postulated to *explain* generalizations—generalizations which can be stated, if not explained so elegantly, with atomic phonemes. For example, a rule which spreads a feature of nasalization across vowels can be expressed as a rule that converts /a/ to /ã/, /o/ to /õ/, /u/ to /ũ/, etc. Phonological rules, and the natural classes used in those rules, can therefore be written in terms of atomic phonemes.

Similarly, while Optimality Theory (the current popular approach to phonology) holds that the phonetic form of words is determined by ranked constraints rather than rules, there is little if any empirical data that cannot be accounted for by a more traditional rule-based approach.[3]

In summary, while we may expect theories of morphology and phonology to continue to evolve, in practice it is quite feasible to do morphology and phonology using today's theories—or even yesterday's theories. Linguistics does not stand in the way of developing a sufficiently strong description language for morphology and phonology, and present-day finite state transducers are capable of implementing these descriptions.

The remainder of this paper sketches a design of a general language for writing morphological grammars, and a method for compiling grammars written in that generic language into the programming language of particular morphological parsers. We also describe how we supplement this formal grammar with a reference grammar of the morphology and phonology of a language. This may be seen as a way of commenting our code, but in fact we argue that it is much more, and that it constitutes a valuable effort in its own right.

[3] The under- and over-application of phonological rules in reduplication is a potential, if rare, counter-example.

2 Interoperability and Half-Life: Solution

We have embarked on a project to build morphological grammars and parsers of languages in a way that overcomes the Interoperability and Half-Life problems described in the previous section. The first grammar we have written is for the Bengali, or Bangla, language.

Our approach is to write a formal grammar in XML, using an XML schema to define the various grammatical structures needed to create—in conjunction with a suitable lexicon—a high coverage morphological parser. At present, this parser is being implemented in the Stuttgart Finite State Transducer (SFST). Since XML is not the native language of this parsing engine, we have written a 'compiler' to convert the XML-based grammar into code which SFST can use.

The following subsections describe our methodology in more detail.

2.1 Descriptive Grammar

During our investigation of Bangla, we were surprised to discover that no thorough and reliable descriptive grammar of modern colloquial Bangla exists, despite its having over 200 million native speakers. Instead, we relied on descriptions of Bangla morphology from half a dozen grammars, several journal articles, and a couple of dissertations. These sources were not always clear, nor did they always agree, and a few fine points of Bangla morphology were simply unexplained.

The difficulty we encountered in understanding grammatical descriptions, reconciling different grammatical accounts, and filling in gaps in coverage underline the fact that we could not have simply picked up an existing grammar and written our formal grammar from that. For languages which have any degree of inflectional complexity—and Bengali does, although there are languages with still more complicated morphologies—the complexities prevent such a simple approach. Instead, we began by writing a descriptive grammar, similar to reference grammars for other languages. It contains a chapter on the phonology and writing system of Bangla, plus chapters for the various parts of speech, describing the inflectional (and some derivational) affixes, and how the resulting inflected forms define the paradigms. The usage of these forms is also described with examples; it is not, however, a pedagogical grammar.

While the formal grammar described below was designed for interoperability, the reference grammar is much more than an add-on; rather, it is the means by which the formal grammar was written.

The following sections describe the formal grammar, and how we combined the descriptive and formal grammars into a unified whole.

2.2 Formal Grammar

In order to produce a morphological parser, one needs an unambiguous description of a language's morphology. Ambiguity is a fact about natural language, and one which plagues software specifications (Berry and Kamsties 2003). Building a parser from a descriptive grammar would be analogous to building traditional software from a specification. The danger is that our reference grammar, like the grammars we consulted, may be unclear or ambiguous, which would prevent its being used ten or a hundred years from now to build a new parser. We therefore need to supplement it with a grammar written in a formal language.

One approach would be to use the programming language of an existing parsing tool as that formal language. Amith and Maxwell (2005) propose using the xfst language (the language of one of the Xerox finite state tools, see Beesley and Karttunen 2003) for archival purposes. While this would meet our need for an unambiguous representation, it would fail to meet our goal of longevity: the Xerox tools will likely not be used in ten years, and there is no reason to think that the morphological parsing engines available then will use the same programming language, or that future grammar engineers will understand the xfst language.

Our formal grammar must therefore be not only unambiguous, but also—as far as possible—iconic and self-documenting. We decided to write our formal grammar in XML, and have developed an XML schema for this purpose.

The XML schema is based on a UML model developed by SIL (downloadable from http://fieldworks.sil.org/). This model allows for a rich set of morphological constructs:

- Item-and-arrangement affixes
- Item-and-process affixation
- Compounding and incorporation
- Paradigm classes, stem allomorphy classes
- A slot-and-template representation for inflectional affixation
- Morphosyntactic features structures
- Exception features
- Allomorphy constraints

Our schema allows for most of these constructs, with the exception for now of item-and-process morphology. We have supplemented the model with ordered phonological rules, which gives us a second mechanism for describing allomorphy.

Our XML schema is intended to "plug and play" with proposed standards for lexical databases, including the ISO draft Lexical Markup Framework (http://lirics.loria.fr/doc_pub/LMF_revision_14.pdf).

While we have built small test cases to exercise specific parts of the model and its schema, building a full-scale grammar allows us to test the schema in other ways. For example, our schema originally called for all regular expressions to be defined in one place, and called by reference (using XML *refid*s) in the various allomorph constraints and phonological rules where they are used. This worked well in small test cases, and it is computationally straightforward; it nevertheless turned out in our Bengali grammar to be too complex for linguists to maintain. As a result, we altered our schema to allow for regular expressions to be either called by reference, or defined where they are used. The former is used for regular expressions which are used often (such as the definition of consonants), while the latter is used for regular expressions that appear only once or twice.

2.3 Combining Descriptive and Formal Grammars

We have, then, both a descriptive and a formal grammar. We have argued elsewhere (Amith and Maxwell 2005a, 2005b) that neither is adequate by itself for long-term language description. We here summarize these claims.

First, we cannot presume that a linguist who was unfamiliar with our XML language could look at our formal grammar and easily deduce what it means. We therefore view our reference grammar as a supplement to the formal grammar. This is like commenting code, except that comments in traditional programs are intended for someone who is already conversant in the programming language,

whereas our reference grammar is intended to explain the meaning of the constructs to someone who does not already understand our XML grammar description language—a more ambitious goal.

On the other hand, since descriptive grammars are written in natural language, they are inherently ambiguous (as discussed above), and even vague. If a formal grammar could be combined with the descriptive grammar, we would have an antidote to this problem; assuming an appropriate syntax, a formal grammar is neither ambiguous nor vague.

The question then is whether the descriptive and formal grammars can be combined, allowing each to make up for the other's deficits. Such a combination would need the following components:

(1) A way to develop the grammars in parallel.
(2) A way to combine the grammars so that the description of each grammar topic is presented to the human reader along with the corresponding rules of the formal grammar.
(3) A way to extract the formal grammar for use by the parsing engine.

In fact, there already is a method that accomplishes (2) and (3); Literate Programming (henceforth LP), developed by Donald Knuth (Knuth 1984, 1992) for documenting computer programs. We have chosen the XML/ DocBook implementation of LP (Walsh and Muellner 1999; Walsh 2002), since XML provides advantages for long-term archiving (cf. the recommendations for the use of XML in Bird and Simons 2002). To this existing framework, we add a development methodology (described in David and Maxwell forthcoming), accomplishing point (1) above.

The result, we hope, is a mechanism that will allow another computational linguist—now or in the future—to pick up our grammar, understand what it means, and convert the formal grammar into the programming language of some other morphological parsing engine, either by writing an automatic converter, or by converting the grammar by hand.

As a reviewer pointed out, Literate Programming has not had a large impact on traditional software engineering. There are however two significant differences which give us reason to believe that LP can be more successful in computational linguistics. First, programmers are engineers, and they are notoriously resistant to writing documentation; and LP puts documentation first, so it is not surprising that software engineers are resistant. Linguistics, on the contrary, has a history of millennia of grammar documentation. Indeed, the problem for many linguists is exactly the opposite: describing grammars is natural, while writing formal language rules is unnatural. So if there is a problem in persuading linguists to do LP, it will be finding linguists who are willing to write the formal grammars (an issue addressed in the abovementioned methodology paper).

We suspect that another reason why software engineers are reluctant to spend the time doing LP is the burgeoning size of many computer programs. Programs which have been documented using LP are typically (if not always) small; but even utility programs, like the familiar command-line utilities of Unix, have become increasingly large and sophisticated. Grammars, on the other hand—at least morphological grammars—are comparatively simple, even for languages with complex morphologies. (Indeed, a grammar which seems too complex is often taken to be incorrect, or at the very least "missing a generalization.") Grammars, it seems to us, are just the right size for LP: not so small that they don't need documenting, and not so large that they cannot be documented. We are thus optimistic about the future of LP grammar writing.

2.4 Conversion to publishable grammar

We view our Bengali grammar, including both the descriptive and formal components, as a publishable work. Style sheets for DocBook of course exist already, giving us publication quality displays for our reference grammar. But while the formal grammar is understandable in its XML form, it is not "pretty" (as evident from the excerpt of our grammar in the appendix), nor does it bear an obvious resemblance to linguistic formalisms.

Fortunately, the flexibility of XML makes it possible to display a formal grammar using linguistic formalisms—for example, using style sheets to convert the XML structures for phonological rules into rules formatted in the way that linguists expect. The creation of the style sheets necessary to display and typeset our formal grammar is planned for next year, giving us the remaining piece needed for the Literate Programming. which Knuth referred to as 'weaving.'

2.5 Conversion to parser

The grammar is also intended to be used by a morphological parser. To build the parser, we first extract the formal grammar as an XML document

from the combined descriptive and formal grammar. This process, known in LP as 'tangling', is done by an XSLT program developed by Norman Walsh (available at http://docbook.sourceforge.net/release/litprog/current/fo/ldocbook.xsl).

The extracted XML formal grammar is then read by a Python program into an internal representation as objects, and output as the programming language of the target morphological parsing engine. A computer-readable lexicon must also be converted into the programming language of the parsing engine—a comparatively simple task.

Finally, the converted grammar and lexicon are read by the parsing engine, currently the Stuttgart Finite State Transducer, to produce the parser.

We expect any choice of parsing engine today to be superseded by more capable parsers. Targeting a different parsing engine will require rewriting only that part of the converter program that translates the program-internal representation into the target programming language (plus a separate converter for the lexicon). Alternatively, for relatively simple grammars it should be possible to translate an XML grammar into the target language by hand, a process aided by the side-by-side exposition of reference and formal grammars provided by the Literate Programming framework.

The analogy to the compilation of high-level programming languages is clear: while we compile our XML language into the high-level programming language of a morphological parsing engine, rather than into the machine language of a CPU, the goal is to make a program usable on a variety of platforms, both now and in the future.

Verifying that the conversion process works correctly with a parsing engine requires test data. Much of this test data can be automatically extracted from the descriptive grammar's paradigm tables and example sentences—another advantage of having both descriptive and formal grammars.

3 Previous work

We are not aware of previous work intended to produce grammars written in a formalism designed to be ported to different parsing engines. The closest work along these lines is perhaps DATR (Evans and Gazdar 1996), a formalism intended for lexical representation, incorporating a general mechanism for non-monotonic inheritance. It is possible to translate a DATR grammar into a morphological parser (see e.g. Colburn 1999). However, DATR is not an XML-based system. Perhaps more importantly, it uses a general purpose inheritance language, whereas our XML schema is a specialized language for morphology, allowing linguists to express linguistic constructs in linguistic terms.

SIL's recent FLEx program (http://www.sil.org/computing/fieldworks/flex/) is a database designed for linguistic field work. It incorporates a parser-independent representation of the morphological grammar; in fact, this is the source of the UML model that we used as the basis of our own XML schema. The SIL design anticipates that different parsers will be used in the future (FLEx currently uses SIL's XAMPLE parser), but the program is designed to support field linguists by providing a particular parser; FLEx was not designed with the goal of making it easy for computational linguists using other parsers to re-use grammars (Black and Simons 2006, Andrew Black, p.c.).

XLingPaper is an XML-based language developed by Andrew Black of SIL to write grammatical descriptions (see http://www.sil.org/~blacka/xlingpap/index.htm). XLingPaper allows for embedding interlinear text into documents, but it does not incorporate a formal grammar.

Some work on models and schemas for lexicons includes partial models of morphology, in particular the previously mentioned ISO draft Lexical Markup Framework (LMF). We are exploring adapting those parts of the 'intensional' and 'extensional' morphology specifications in LMF which overlap with our model. To some extent, LMF has been a moving target, and merging the two models will require further effort. Also, the ISO standard for feature structures (ISO 24610-1:2006) is used in our morphology model to represent morphosyntactic features.

There is also considerable work by computational linguists on defining models and schemas for lexicons and annotated text. We see our work as extending this to grammar development.

4 Conclusion

What is new about the project we describe is the development of an XML schema based on a model of morphology and phonology, and intended as a way of developing and documenting grammars so that they are easily ported to morphological parsing engines, both present and future.

While it is not a necessary part of modeling grammars for parser building, we believe that combining the formal XML-based grammar with a reference grammar intended to be read by humans provides increased portability. The combination serves as a better form of archival language documentation and description than either the reference grammar or the formal grammar by itself would.

Finally, we note that while our focus has been on morphological grammars, similar techniques—the development of a generic model, and the use of Literate Programming—could be applied to syntax. There is however perhaps less agreement on appropriate models among syntacticians than there is among morphologists, making this more of a hope than an immediately achievable goal.

References

Amith, Jonathan D., and Maxwell, Michael. 2005. Language Documentation: The Nahuatl Grammar. In Alexander Gelbuck (ed.) *Computational Linguistics and Intelligent Text Processing*. Lecture Notes in Computer Science. 474-485. Berlin: Springer.

Antworth, Evan L. 1990. *PC-KIMMO: a two-level processor for morphological analysis.* Occasional Publications in Academic Computing No. 16. Dallas, TX: Summer Institute of Linguistics.

Beesley, Kenneth R., and Karttunen, Lauri. 2003. *Finite State Morphology*: CSLI Studies in Computational Linguistics. Chicago: University of Chicago Press.

Berry, Daniel M., and Kamsties, Erik. 2003. "Ambiguity in Requirements Specification." In Julio Cesar Sampaio do Prado Leite and Jorge Horacio Doorn (eds.) *Perspectives on Software Requirements*. The Springer International Series in Engineering and Computer Science. Vol. 753. Berlin: Springer.

Bird, Steven, and Simons, Gary. 2002. Seven Dimensions of Portability for Language Documentation and Description. In *Proceedings of the Workshop on Portability Issues in Human Language Technologies, Third International Conference on Language Resources and Evaluation*. Paris: European Language Resources Association.

Bird, Steven, and Simons, Gary. 2003. Seven dimensions of portability for language documentation and description. *Language* 79:557-582.

Black, H. Andrew, and Gary F. Simons. 2006. "The SIL FieldWorks Language Explorer Approach to Morphological Parsing." *Computational Linguistics for Less-studied Languages: Proceedings of Texas Linguistics Society*, Austin, TX. http://www.sil.org/~simonsg/preprint/FLExParser%20Preprint.pdf

Butt, Myriam, King, Tracy Holloway, Niño, María-Eugenia, and Segond, Frédérique. 1999. *A Grammar Writer's Cookbook*: CSLI Lecture Notes, 95. Stanford, CA: CSLI Publications.

Colburn, Michael. 1999. "Enabling a Legacy Morphological Parser to use DATR-based Lexicons." Ph.D. dissertation, Colorado Technical University. http://ogea.org/Linguistics/Colburn2000.pdf.

Copestake, Ann, and Flickinger, Dan. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece.

Creutz, Mathias, and Lagus, Krista. 2007. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing* 4.

Cunningham, H., Tablan, V., Bontcheva, K., and Dimitrov, M. 2002. Language engineering tools for collaborative corpus annotation. http://citeseer.ist.psu.edu/734322.html.

David, Anne, and Michael Maxwell. Forthcoming. "Joint Grammar Development by Linguists and Computer Scientists." *Workshop on NLP for Less Privileged Languages, IJCNLP 2008*. Hyderabad.

Evans, R. and Gazdar, G. 1996. DATR: a language for lexical knowledge representation. *Computational Linguistics* 22: 167-216.

Goldsmith, John. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics* 27:153-198.

Goldsmith, John , and Hu, Yu. 2004. From Signatures to Finite State Automata. *Midwest Computational Linguistics Colloquium*, Bloomington IN.

Knuth, Donald E. 1984. Literate programming. *The Computer Journal* 27:97-111.

Knuth, Donald E. 1992. *Literate Programming*: CSLI Lecture Notes. Stanford: Center for the Study of Language and Information.

Ma, Xiaoyi, Lee, Haejoong, Bird, Steven, and Maeda, Kazuaki. 2002. Models and Tools for Collaborative Annotation. In *Proceedings of the Third International Conference on Language Resources and Evaluation*. Paris: European Language Resources Association.

Maxwell, Michael B. 2002. *Proceedings of the Workshop on Morphological and Phonological Learning*. New Brunswick, NJ: ACL.

Nirenburg, Sergei, Biatov, Konstantin, Farwell, David, Helmreich, Stephen, McShane, Marjorie, Ponsford, Dan, Raskin, Victor, and Sheremetyeva, Svetlana. 1999. Toward Descriptive Computational Linguistics. http://crl.nmsu.edu/expedition/publications/boas-acl99.pdf.

Oepen, Stephan, Flickinger, Dan, Tsujii, Jun-ichi, and Uszkoreit, Hans. 2001. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*: CSLI Lecture Notes, 118. Chicago: University of Chicago Press.

Oflazer, Kemal, Nirenburg, Sergei, and McShane, Marjorie. 2001. Bootstrapping Morphological Analyzers by Combining Human Elicitation and Machine Learning. *Computational Linguistics* 27:59-85.

Walsh, Norman, and Muellner, Leonard. 1999. *DocBook: The Definitive Guide*. Sebastopol, California: O'Reilly & Associates, Inc.

Walsh, Norman. 2002. Literate Programming in XML. *XML 2002*, Baltimore, MD.

Weber, David; Andrew Black; and Stephen R. McConnel. 1988. *AMPLE: A tool for exploring morphology*. Occasional Publications in Academic Computing no. 12. Dallas: Summer Institute of Linguistics.

Appendix: Sample Grammar Excerpt

3.2. Future Tense

The future tense is used to express:

- a future state or action
- propriety or ability [*etc.*]

…

Table 6.2. FutureTense Verb Forms

Person	Suffix	(C)VC-	(C)aC-	(C)V-	(C)a-	(C)V(i)-	Causative	3-অক্ষর্
		শোনা /ʃon-a/ *to hear*	থাকা /thak-a/ *to stay*	হওয়া /hɔ-oya/ *to become*	খাওয়া /kha-oya/ *to eat*	চাওয়া /ca-oya/ *to want*	শেখানো /ʃekha-no/ *to teach*	কাম্‌ড়ানো /kamṛa-no/ *to bite*
1st	-বো /-bo/	শুন্‌বো /ʃun-bo/	থাক্‌বো /thak-bo/	হব /hɔ-bo/	খাব /kha-bo/	চাইব /cai-bo/	শেখাব /ʃekha-bo/	কাম্‌ড়াব /kamṛa-bo/

[*Additional rows omitted to save space*]

The formal grammar's listing of future tense suffixes appears below.

```
<Mo:InflectionalAffix gloss="-1Fut" id="af1Fut">
   <!--The two "allomorphs" are really allographs-->
   <Mo:Allomorph form="বো">
      <!--Spelled 'bo'; usually (not always) after a C-stem -->
   </Mo:Allomorph>
   <Mo:Allomorph form="ব">
      <!--Spelled 'b'; usually (not always) after a vowel stem -->
   </Mo:Allomorph>
   <Mo:inflectionFeatures>
     <Fs:f name="Tense"><Fs:symbol value="Future"/></Fs:f>
     <Fs:f name="Mood"><Fs:symbol value="Indicative"/></Fs:f>
     <Fs:f name="Person"><Fs:symbol value="1"/></Fs:f>
   </Mo:inflectionFeatures>
/Mo:InflectionalAffix>

<!-- Etc. for the remaining future tense suffixes -->
```