Joint Grammar Development by Linguists and Computer Scientists

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 inflectional morphology, development of a morphological parser can be a bottleneck to further development. We focus on two difficulties: first, finding people with expertise in both computer programming and the linguistics of a particular language, and second, the short lifetime of software such as parsers. We describe a methodology to split parser building into two tasks: descriptive grammar development, and formal grammar development. 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, so that it can be readily ported to a new parsing engine, thus helping solve the software lifetime problem.

1 Problems for Grammar Development

After several decades of widespread effort in computational linguistics, the vast majority of the world's languages lack significant computational resources. For many languages, this is attributable to the lack of even more basic resources, such as standardized writing systems or dictionaries. But even for many languages that have been written for centuries, computational resources are scarce.

One resource that is needed for languages with significant inflectional morphology is a morphological parser.[1] To the degree that a language has complex morphology, parsers are difficult to build. While there has been considerable research into automatically deriving a morphological parser from a corpus (see for example Creutz and Lagus, 2007; Goldsmith, 2001; Goldsmith and Hu, 2004; and the papers in Maxwell, 2002), the results are still far from producing reliable, wide-coverage parsers. Hence most morphological parsers are still built by hand. This paper focuses on practical aspects of how such parsers can best be built, and presents a model for collaborative development.

Hand-built parsers suffer from at least two drawbacks, which we will call the 'Expertise Problem' and the 'Half-Life Problem.' The Expertise Problem concerns a difficulty for building a parser in the first place: it is hard to find one person with the necessary knowledge of both the linguistics of the target language and the computational technology for building parsers.

The Half-Life Problem concerns the fact that once a parser has been built, its life is limited by the life of the software it has been implemented in, and this lifetime is often short.

The following subsections further describe these two problems, while the remainder of the paper focuses largely on the Expertise Problem. We focus specifically on the development of morphological grammars. The techniques described here may be usable with syntactic grammars as well, but we have not investigated that problem. We also focus in this paper on the development issue; testing and debugging grammars is not discussed in this paper.

1.1 The Expertise Problem

Writing software requires two kinds of expertise: knowledge of the problem to be solved, and knowledge of how to program software. For parsers, the problem-specific knowledge requires understanding the grammar of the target language. Since everyone speaks at least one language, it

[1] In fact, it is more common to create 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 frequently use the term 'parser.'

Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages, pages 27–34,
Hyderabad, India, January 2008. ©2008 Asian Federation of Natural Language Processing

might seem that finding someone who understands the grammar of any particular language should be easy. Unfortunately, as generations of field linguists have discovered, this is not true. A native speaker's knowledge of a language is notoriously implicit; converting that knowledge into explicit rules is no simple task. Furthermore, finding a speaker of the language who combines explicit understanding of the grammar with software engineering skills is even more difficult. The difficulty is compounded when the number of speakers of the language is small. We therefore believe that for many languages of the world, for the near future, the way to develop computational tools in general, and morphological parsers in particular, lies in teamwork.

An example of the team approach was the BOAS project (Oflazer et al., 2001). A BOAS team consisted of two people—a 'language informant' and a programmer—plus a computer program which interviewed the informant and created the grammar rules. The computer program is described as a 'linguist in a box' (Oflazer et al., 61). The method we describe uses computational tools, but purely human teamwork.

A potential problem with the team approach lies in facilitating communication between team members. While electronic communication makes distributed teams possible, there is still a question of how best to enable people with disparate skills to actually understand each other. We return to this below, when we discuss our collaborative method.

1.2 The Half-Life Problem

Another problem with computational tools is their lack of longevity. While it would be difficult to formally investigate, we estimate the average lifetime for computational linguistic tools to be five or ten years. In part, this is due to the (lack of) longevity of the underlying software.[2] Of course, some vendors provide backwards compatibility, and not all software becomes extinct that quickly—but that is the meaning of 'half-life.'

Software obsolescence can be postponed by the judicious choice of programming languages, avoiding platform- or OS-specific commands, the use of open source methods, etc. However, this can only prolong the life of a program, not extend it indefinitely.[3] There are few if any programs that were written in 1980 that still run on any but computers outside of a museum—and 1980 was only twenty-seven years ago.

In contrast, natural languages change slowly, apart from the infusion of new vocabulary. The grammar of a language spoken today is unlikely to be significantly different from the grammar of that same language fifty or a hundred years ago; and barring catastrophe, any changes which do happen are likely to be incremental.

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

Another motivation for wanting to build parsing tools with a longer half-life is that they constitute a description of (part of) the grammar of a language, in two senses: first, the grammar that the parser uses is in effect a formal description of the language's morphology (or syntax). This formal description has the advantage over traditional grammar descriptions of being unambiguous.

A second way in which a parser constitutes documentation of a language is that 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 a static one.

However, linguists have drawn attention to the issue of longevity for computer-based language documentation and description. In their seminal paper, Bird and Simons (2003) point out that the use of digital technologies brings the potential that archive language data can become unusable much more quickly than printed grammatical descriptions. Indeed, scholars of today can understand grammars of South Asian languages penned thousands of years ago.

[2] One of us (Maxwell) was involved in a project in which two of the programming languages became defunct before the program was complete. In both cases, the cost of porting to alternative dialects of the programming language was deemed prohibitive.

[3] Old software can of course be kept on "life support" by running it on old machines running old operating systems. But that is a solution for museums, not for software that is intended to be actively used.

Since a parser embodies a description of the grammar of a language, it should be written to provide an explicit, computationally implementable description of the language, portable to future parsing engines even after the language is extinct. As we show below, this is not an impossible goal.

2 A method for Grammar Development

We have embarked on a project to build morphological parsers of languages in a way that overcomes the Expertise and Half-Life problems described in the previous section. The first parser was for the Bengali, or Bangla, language. Our choice of Bangla was driven by a number of considerations, many of which are not relevant here. Most any language with a significant amount of inflectional morphology would have worked. However, in retrospect the choice was a good one, as it forced us to deal with a number of both computational and linguistic issues that a more highly resourced language such as Spanish would not have presented. At the same time, Bangla is sufficiently documented by traditional grammars that the task was achievable, although not as easy as we had anticipated.

We are writing two kinds of grammars simultaneously: the first is a traditional descriptive or reference grammar, written in English prose by a linguist (Anne David), intended to be read by linguists. The other is a formal grammar, written in a formal specification language, by a computational linguist (Mike Maxwell) and intended for conversion into the programming language of a parsing engine. (Neither of us is a speaker of the Bangla language.) The two grammars are intertwined, as described below, so that each supports the other in such a way that we can combine our differing expertise while also avoiding the lack of longevity that plagues traditional parser development.

The following subsections describe the methodology we are using, and its advantages.

2.1 Descriptive Grammar

The descriptive grammar we have written is not, of itself, ground-breaking. Like most reference grammars of the morphology of a language, it has a chapter on the phonology and writing system of Bangla, and chapters for the various parts of speech. The latter chapters describe the inflectional (and some derivational) affixes each part of speech takes, and how the resulting inflected forms define the paradigms. The usage of these forms is also described, with examples sufficient to illustrate the usage; it is not, however, a pedagogical grammar.

We were surprised to discover that no thorough and reliable English-language descriptive grammar of modern colloquial Bangla exists, despite its having well over 200 million native speakers. Instead, we had to glean our description of Bangla morphology from half a dozen or so grammars of varying quality (some of them pedagogical[4]), several journal articles, and a couple of dissertations. Doing so meant comparing and reconciling sometimes widely differing descriptions and analyses; three major problems we encountered were contradictory accounts, lack of clarity, and gaps in coverage. Writing a formal grammar forced us to both resolve these issues and clarify our descriptive grammar.

For example, we knew from our sources that the locative/ instrumental case in Bangla has several allomorphs; however, the descriptions of their distribution differed, and one of our chief sources was, in fact, quite vague on the conditioning environments. Moreover, one particular vowel alternation that takes place in certain verb forms goes unmentioned in nearly all of our sources and is inaccurately described in one of the two that do mention it. In this instance, a native speaker confirmed the correct forms for us. Opinions among the written sources on how to classify Bangla verbs differed widely as well, with anywhere from two to seven classes proposed. We ended up choosing the system that defined seven stem classes, since it is the only one that enables the generation of any verb form, given a stem.

Resolving such problems was made easier by the help of a consultant in the Bangla language. Professor Clint Seely, Emeritus of the University of Chicago. He corrected our many mistakes and helped clear up ambiguities in our sources.

The difficulties 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 a grammar and

[4] In fact, the clearest and most reliable sources of information were pedagogical grammars.

written a formal grammar from it. For languages which have any degree of inflectional complexity—and Bengali does, although there are languages with still more complicated morphologies—the problems are too great for such a simple approach. One might ask why it is so difficult to convert a published grammar into a morphological parser. One answer is that languages are inherently complex. It is common for published descriptions to overlook complexity, either in the interest of presenting a simple and general description, or perhaps because the author is unaware of some of the issues.

Also, as any reader or writer of technical papers knows, it is all too easy to talk about complex topics unclearly. In our case, writing the formal grammar at the same time as the descriptive grammar forced a clarity and breadth of coverage in our descriptive grammar which we would not otherwise have attained. Moreover, by incorporating a formal grammar into the descriptive grammar, we have gone beyond previous work on Bangla, or most other languages. The following section describes this.

2.2 Formal Grammar

For the formal grammar of Bangla morphology, we need a description which is unambiguous and capable of being used to build a morphological parser. As discussed above, ambiguity is a fact about natural language, and one which has long plagued software specification efforts (Berry and Kamsties, 2003). Building a parser from a descriptive grammar is analogous to building traditional software from a software specification.

Since our descriptive grammar is a natural language specification, it is *not* what an implementer would want to rely on. We therefore needed a formal language for grammar writing.

One approach would be to use the programming language of an existing parsing tool. Amith and Maxwell (2005a) propose using the xfst language (the language of one of the Xerox finite state tools, see Beesley and Karttunen, 2003). While this would meet the 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 whatever morphological parsing engines are available then will use the same programming language—nor that grammar engineers will understand the xfst programming language.

Our formal grammar needs to be unambiguous, iconic, and self-documenting. We have therefore chosen to represent our formal grammar in XML, and have developed an XML schema for encoding linguistic structures, based on a UML model developed by SIL researchers.[5] The design goals of our XML schema are described in more detail in Maxwell and David (forthcoming).

2.3 Combining Descriptive and Formal Grammars

However, as we have argued elsewhere (Amith and Maxwell, 2005a; 2005b), neither a descriptive nor a formal grammar is adequate to our purposes by itself. Descriptive grammars are inherently ambiguous and sometimes vague, while formal grammars are hard to understand. If a formal grammar could be combined with the descriptive grammar, we would have an antidote to these problems: the combination could be neither ambiguous nor vague.

The question is then whether there is a way to combine the two sorts of grammars. Such a method would need to support the following:

(1) Developing the grammars in parallel.
(2) Combining the grammars so that the description of each aspect of the grammar is presented to the human reader along with the corresponding aspect of the formal grammar.
(3) Extracting the formal grammar for use by the parsing engine.

In fact, there already is a method that accomplishes (2) and (3): Literate Programming, developed by Donald Knuth (1984, 1992) as a way of documenting computer programs. We use an XML/ DocBook implementation of Literate Programming (Walsh and Muellner, 1999; Walsh, 2002), since XML provides numerous advantages for long-term archiving (cf. Bird and Simons, 2002).

There remains the need for a methodology for developing the descriptive and formal grammars in parallel, point (1) in the above list. We turn to this question in the next section.

[5] The SIL model can be downloaded from http://fieldworks.sil.org/.

2.4 Collaborative Grammar Development

We are writing our descriptive grammar of Bangla in a commercial program, XMLmind (http://www.xmlmind.com/xmleditor/). The formal grammar is being written in a programmer's editor, although with suitable style sheets, it could be written in XMLmind. The formal grammar consists of a number of 'fragments,' each paired with a section in the descriptive grammar, so that the descriptive and formal grammatical descriptions are mutually supportive (see the appendix for a short excerpt).

Our working arrangement is one of iterative development, with descriptive grammar writing leading formal grammar writing. Crucially, this iterative development allows frequent exchanges for clarification. A typical interchange (one which actually took place) is the following. The language expert writes a section of the descriptive grammar on Bengali noun qualifiers. The computational grammar writer reads the description and tries to implement it, but a question arises: is the diminutive qualifier used in all the environments that the three allomorphs of the non-diminutive qualifier are used, or only one of those environments? The language expert finds examples showing the diminutive in all environments, enabling the computational grammar writer to proceed. Crucially, the descriptive grammar was then modified to clarify this issue, and to include the new examples.

Although we are writing our grammars a short hallway apart, this interchange was accomplished largely by email; we could as well have been a continent apart.

In summary, our division of labor, together with the fact that we are simultaneously developing the two kinds of grammar using our computational tools and incorporating immediate feedback, has made possible a much better result than if one of us wrote the descriptive grammar, and the other later wrote the formal grammar.

2.5 Conversion to publishable grammar

As evident from the small portion of our grammar in the appendix, the formal grammar is understandable in its XML form, but it is not "pretty"; nor does it bear any obvious resemblance to modern linguistic formalisms.[6] At the same time, the use of XML means that a variety of tools are available for editing the grammar, checking its validity against the schema, and converting it into the programming language of a parsing engine.

Fortunately, the flexibility of XML makes it possible to display (and eventually publish) the formal grammar using linguistic formalisms, such as the following:

$$\left\{\begin{matrix} \text{p} \\ \text{t} \\ \text{k} \end{matrix}\right\} \rightarrow \left\{\begin{matrix} \text{p}^{\text{h}} \\ \text{t}^{\text{h}} \\ \text{k}^{\text{h}} \end{matrix}\right\} \quad / \, \# \underline{\quad} \text{V}$$

The ability to create such display forms of the underlying XML data—referred to by Knuth as "weaving"–is important as we look to publishing the combined descriptive and formal grammar. The creation of the style sheets necessary for this is planned for next year.

2.6 Conversion to parser

To build a parser from our grammar, we first extract the formal grammar as an XML document from the combined descriptive and formal grammar. This is a standard process in Literate Programming, called 'tangling'; we use a simple XSLT (Extensible Stylesheet Language Transformation), developed by Norman Walsh (http://docbook.sourceforge.net/release/litprog/current/fo/ldocbook.xsl).

Second, the extracted XML formal grammar is read by a small Python program, then converted into 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 to produce the parser. Currently, the target parsing engine is the Stuttgart Finite State Transducer Tools (http://www.ims.uni-

[6] We have resisted the temptation to make our linguistics too modern, since linguistic theories also have a short half-life. We model an eclectic but largely 1950s era version of linguistics. For example, phonological natural classes are defined by listing the phonemes of which they are composed, rather than using distinctive features; we use ordered phonological rules, rather than Optimality Theory-style constraints rankings. While these may be outmoded, they are quite understandable.

stuttgart.de/projekte/gramotron/SOFTWARE/ SFST.html). We fully expect that any choice of parsing engine we make today will be superseded in the future by better and more capable parsing engines. Targeting a different parsing engine will require rewriting only that part of the conversion program that re-writes the program-internal representation into the target programming language (plus a converter for the lexicon).

Verifying that the conversion process works correctly with a new parsing engine will require standard test data. Much of this test data can be automatically extracted from the paradigm tables and example sentences of the descriptive grammar.

3 Previous work

Collaborative work on natural language processing programs is not of itself a new idea. It is quite common to split up the task of developing a grammar among people with skills in linguistics, lexicography, and software development. In that sense, our work is very traditional.

Ours is not even the first effort at developing a framework for collaborative development of computational linguistic tools. Butt et al. (1999) describe the development of grammars in several languages, including English, French and German (with other languages added later). However, their focus was on enabling collaboration among grammar writers working in different languages; each author was assumed to be more or less skilled in one target language and in computational linguistics. Their focus thus differs from ours in its scope and in the nature of the collaboration.

Copestake and Flickinger (2000) devote a section to "Collaborative grammar coding," but conclude that in order to work on a (syntactic) parser, a developer needs to combine skills in the linguistic theory being implemented, grammar debugging, and the grammar of the target language. In our work, we are attempting to make it possible to split this expertise between different people, and to provide them with a collaborative tool.

Significant effort has been directed at enabling collaborative annotation of corpora, e.g. Cunningham et al. 2002, and Ma et al. 2002. This is similar to our approach in allowing collaboration between annotators and experts (annotation supervisors); but unlike our project, collaborative annotation does not address grammar development.

Finally, there are linguistic development environments such as SIL's FLEx (www.sil.org/computing/fieldworks/flex/), and the planned Montage project (Bender et al., 2004), which are intended to help linguists write computational grammars, incorporating or generating descriptive grammars. While these are useful tools—we are in fact looking into using FLEx to produce interlinear sentences for our grammars—they are not intended for the same kind of collaborative effort that we describe here.

4 Conclusion

What is new about the project we describe is therefore the development of a computational framework within which computationally implemented grammar development can be split into distinct tasks: one task for a person (or a team) with knowledge of a particular language, and another task for a person (or team) with skills in computer science. (Lexicography may constitute a third task, depending on whether suitable machine-readable dictionaries are already available.)

If this division of labor we describe here were applicable only to the working relationship between the authors, it would be of little general interest. However, we believe a similar division of skills between language expert and computational expert to be quite commonplace, making the same division of labor workable in a variety of scenarios. This has implications for the development of linguistic software in low density languages: finding someone who is expert in both a language and its grammar, and in computational techniques, is likely to be particularly difficult in the case of languages which have not been well-documented, or minority languages, or languages spoken in countries where there is not a history of work in natural language processing.

It is easy to imagine other scenarios where this division of labor would work. For example, the linguistic team might be part of the language or linguistics department of a university, while the computational team might be part of a computer science department. Grammar development could easily be an open source project, with the developers never meeting face-to-face.

A question which occurred to us many times during this project is, who can best build a grammar or parser for a language: people like us, who are linguists but do not know the language, or native speakers of the language? The answer is not at all obvious. We suggest that the answer is neither one—alone. None of the language speakers or researchers we talked with in the course of this project had the expertise to build and test formal grammars or morphological parsers. At the same time, when the grammars we consulted were not clear, or contradicted each other, we needed to consult with native speakers or researchers to determine the correct answers.

Hence, we feel strongly that parsers and grammars should be built by teams including people with a variety of skills. Given modern technology, it seems clear that the division of labor which our method allows means that there is no reason the people involved in the project need even be in the same country, or all speak the target language.

In sum, we are developing a methodology to build certain kinds of NLP resources in lower density languages, and we have demonstrated this technology for morphological parsing.

References

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

Amith, Jonathan D., and Maxwell, Michael. 2005b. “Language Documentation: Archiving Grammars.” *Chicago Linguistic Society* 41.

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

Bender, Emily M.; Dan Flickinger; Jeff Good; and Ivan A. Sag. 2004. “Montage: Leveraging Advances in Grammar Engineering, Linguistic Ontologies, and Mark-up for the Documentation of Underdescribed Languages.” *Proceedings of the Workshop on First Steps for Language Documentation of Minority Languages: Computational Linguistic Tools for Morphology, Lexicon and Corpus Compilation, LREC 2004*.

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.

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.

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.

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.

Maxwell, Michael B., and Anne David. Forthcoming. “Interoperable Grammars.” Paper to be presented at *The First International Conference on Global Interoperability for Language Resources* (ICGL 2008), Hong Kong.

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.

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.

Appendix: Sample Grammar Excerpt

3.2. Future Tense

The future tense is used to express:

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

…

Person	**Suffix**	**(C)VC-**	**(C)aC-**	**(C)V-**	**(C)a-**	**(C)V(i)-**	**Causative**	**3-**
		/ on-a/ *to hear*	/thak-a/ *to stay*	/h -oya/ *to become*	/kha-oya/ *to eat*	/ca-oya/ *to want*	/ ekha-no/ *to teach*	/kam a-no/ *to bite*
1st	- /-bo/	[illegible]	[illegible]	/h -bo/	/kha-bo/	/cai-bo/	/ ekha-bo/	[illegible]

Table 6.2: FutureTense Verb Forms

[*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="   ">
      <!--Spelled 'bo'; usually (not always) after a C-stem -->
   </Mo:Allomorph>
   <Mo:Allomorph form=" ">
      <!--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 -->
```