From Legacy Lexicon to Archivable Resource

Mike Maxwell

Linguistic Data Consortium
3600 Market St, Suite 810
Philadelphia, PA 19104 USA
maxwell@ldc.upenn.edu

Abstract

A common format for lexicons produced in field linguistics projects uses a markup code before each field. The end of each field is implicit, being represented by the markup code for the next field. This markup format, commonly called "Standard Format Code(s)" (SFM), is used in one of the most common lexicography tools used by field linguists, Shoebox. While this plain text format satisfies many of the desiderata for archival storage of language materials (as outlined in Bird and Simons 2003), there are usually problems with such lexicons as they are produced in practice which detract from their value.

In particular, SFM-coded lexicons commonly suffer from inconsistencies in the markup codes, especially in terms of the adherence of the fields to a hierarchical order (including omission of fields required by the presence of other fields). It is also common for the contents of certain fields to be limited to a fixed set of items, but for the lexicographer to have been inconsistent in the spelling of some of those items. Finally, spell checking (and correction) needs to be carried out in various languages, including both the glossing language(s) and the target language (where possible).

This paper outlines some tools for correcting these problems in SFM-coded lexicons.

Introduction: The Problem

One of the most important results of a typical field linguistic program is a bilingual dictionary. Most dictionaries are prepared in electronic format, often in the flat text format. Other formats can generally be converted into a plain text format.

At present, the most common flat file format is that produced by the SIL program Shoebox. This format utilizes a markup code at the beginning of each field; often this code begins with a backslash, e.g. "\w " for a headword. These tags are therefore known as "backslash markers", or more formally as "Standard Format Markers" (SFMs).

The end of a field is only marked implicitly, by the SFM of the following field. (Fields may occupy more than one line; normally, newlines within fields have no meaning.)

The beginning of a record is marked by the presence of a designated SFM (often either that of the headword field or an arbitrary record number, so that the designated SFM performs a dual function as field marker and record marker). The end of a record is marked by the beginning of the next record. (Often there is a blank line separating records, but this is neither sufficient nor necessary.)

While plain text format satisfies many of the desiderata for archival storage of language materials (as outlined in Bird and Simons 2003), there are certain typical problems with such SFM-coded lexicons as they are produced in practice which detract from their value.

One such problem has to do with the fact that dictionaries are actually structured objects, with logical constraints on the structure of fields within a record (lexical entry), the relationships between lexical entries, and on the contents of the fields themselves. While the structure can be represented using appropriate markup, in practice field linguists' lexicons violate the constraints, both at the level of the markup and at the level of the contents of the fields.

LinguaLinks (another SIL program) has a built-in model of lexical entries which enables it to impose well-formedness constraints at data entry time. However, LinguaLinks does not enjoy the large market share among field linguists that Shoebox does. While it is possible to impose some constraints on a Shoebox dictionary at data entry time, it is possible to do more validity checking in batch mode, provided there is a model of the lexicon. Such a model implies making explicit the semantics of the fields, a semantics which is implicit (albeit sometimes imperfectly so) in the user's mind when he (or someone else) designed the database.[1]

The purpose of this paper is to demonstrate the use of automatic validity checkers which can be applied off-line to an SFM-coded lexicon, marking up the database for errors in batch mode; the errors can then be searched for and corrected on-line. These checkers (or their predecessors) have proven useful in practice on a variety of text-based lexicons. The checks performed include:

- Verifying the markup codes, including their relative ordering and hierarchy (as specified by a model);
- Listing the parts of speech and other restricted fields, with occurrence counts (useful for finding erroneous field content);
- Doing spell for data in languages for which a spell checker is available, and character n-gram checking for languages for which no spell checker is available.

[1] There are in fact several well thought-out models of lexical databases which could be applied to the problem. Generally these models are hierarchical (e.g. senses within lexical entries), but they usually allow for cross-references as well (e.g. synonymy relations, major-minor lexical entries). This is well-suited to an XML structure. Unfortunately, while Shoebox has an XML export capability, it does not create a DTD or Schema, and there are some problems with its XML export (see e.g. http://www-nlp.stanford.edu/kirrkirr/dictionaries/).

In addition, I demonstrate a way to export the lexicon to Microsoft Word format, automatically marking the fields for their language so that the multi-lingual spell correction tools of Word can be applied.

There are other consistency checks which could also be performed. Shoebox has the built-in capability of checking that for every cross-reference, the target of that cross-reference exists. However, Chris Manning (p.c.) has suggested that one should also check for bidirectional references (e.g. synonyms), and this checking capability is not built into Shoebox. This sort of check may be added to the suite of tools described here in the future. Another useful check would be that sense numbers begin with 1 and are sequential.

The validity checking tools will be made available at a public website.

Verifying Markup Codes

Version 5 of Shoebox[2] provides a number of checks that can help ensure consistency. Hence, while it is not necessary to use Shoebox to maintain an SFM-coded dictionary (or other database), Shoebox is a useful tool in the verification process.

Most of the consistency checks in Shoebox are set up using Shoebox's "Database Type Properties" dialog box. For example, Shoebox can be told which fields can be empty, and it will check for fields which should be filled, prompting the user to fill in the missing data. However, while Shoebox can be told which field should follow a given field, it only uses this information when the user adds a new field[3]; it does not check for missing fields which should follow a given field in existing data.

Hence, the first consistency check described here ensures that all required fields are present. It would be helpful if the information concerning the fields could be extracted from the dictionary's 'type' file.[4]

An example of the information in one record of the .typ file appears here:

```
\+mkr d
\nam Definition (English)
\lng English
\MustHaveData
\mkrOverThis w
\mkrFollowingThis dfr
\-mkr
```

The field labeled \mkrOverThis defines the parent SFM of the given SFM: in this case, a \d field appears under a \w field. Unfortunately, this is not sufficient to describe the notion of an obligatory field. That is, the presence of a given field implies the presence of its hierarchical parent, and the presence of an immediately following field (if any). But there is no way to encode the necessity for a field which must appear, but which may not appear immediately after a given field. For example, if a record must have a definition field following a part of speech field, but a usage comment may optionally intervene, there is no way to encode this in the .typ file.

Accordingly, the consistency check for required fields must use its own representation of the dictionary structure. It therefore employs a standard regular expression notation to encode both the hierarchy and the obligatoriness of field structure within records, and the record structure within a dictionary file.[5] The following is an example expression defining the field structure of a dictionary file (the full notation is given in the program documentation):

```
id
(w
   ( (pos defn  (ex exEn exFr)*  (syn)?)
   | (num  pos  defn (ex exEn exFr)*  (syn)?)+
   )
)+
```

This is interpreted as follows. A dictionary file begins with a single \id record. Each following record is marked by a \w field, and may contain either of two alternatives: One alternative contains a part of speech (\pos), definition (\def), zero or more example sentences (\ex), each of which must have both an English (\exEn) and a French (\exFr) translation), and an optional cross-reference to a synonym (\syn; the optionality is indicated by the question mark). The other alternative consists of a one or more senses (represented implicitly), each of which contains a sense number (\num), followed by the same contents as the first alternative.[6]

Notice that the topmost structure is defined at the level of a dictionary file, not the entire dictionary. For many dictionaries, no such distinction is relevant: the entire dictionary is contained within a single file. It is not

[2] All remaining references will be to version 5 of Shoebox. More recently a similar tool called 'Toolbox' has been released (see http://www.sil.org/computing/catalog/show_software.asp?id=79) I have not tested the techniques in this paper under Toolbox, however Toolbox claims to be upwards compatible from Shoebox, so the procedures should work. Toolbox also includes a verification mode for glossed interlinear text, a feature of earlier versions of Shoebox which was omitted from version 5.

[3] In fact, a required field is only added when one hits the Enter key after adding the parent field of the required field. For example, adding an example sentence field will not add a field for the translation of that example sentence until the user hits the enter key at the end of the example sentence. Users may not in fact hit the Enter key when adding fields, so missing fields can arise even after the hierarchy of fields have been established.

[4] The name and location of the type file is given in the Database Types dialog box (Projects menu), and is created by Shoebox from the information in the previously referred-to Database Type Properties dialog box. The latter should therefore be checked for accuracy. Since Shoebox builds the information in that dialog from the database itself, it may contain obsolete information (e.g. SFMs which were used in earlier stages of the work). An undocumented feature is that only those SFMs which are actually used in the dictionary appear in bold in the Database Type Properties dialog. In most cases, any non-bold markers should therefore be removed.

[5] Allowing alternative record structures within the lexicon allows for different kinds of entries, such as minor entries. It also allows for various bookkeeping records that Shoebox includes, primarily at the top of the file.

[6] There is obvious redundancy in this description, which could be eliminated by use of something like the Backus Naur Form. For the sake of readability, I have not employed such a notation.

uncommon, however, for larger dictionaries to be maintained in separate files. For purposes of field checking, however, it should be sufficient to process each such file separately, since records should not cross file boundaries.

The operation of the field checker is as follows: it first reads in the regular expression defining the lexicon structure. It then reads a lexicon file in. Following the SFM notation, records are assumed to be everything from the record-marking SFM in one record to the next record-marking SFM, or to the end of the file (where a record-marking SFM is any top-level SFM in the regular expression). Ambiguity is unlikely here, but the parser uses an anti-greedy algorithm: the first SFM which could begin a new record is assumed to do so. All fields encountered before the next record-marking SFM are assigned to the current record.

Within a single record, the checker then attempts to assign the field markers actually found to the expected field structure. In case of error, a fall-back algorithm is used which allows for the possibility of an inappropriately missing field. For instance, suppose the parser encounters the following structure:

```
\ex    Yax bo'on ta sna Antonio.
\exEn I'm going to Antonio's house.
\ex    Ban yax ba'at?
\exEn Where are you going?
\exFr  Ou allez-vous?
```

Given the field definition above, there is a missing \exFr field after the first \exEn field. The parser encounters the second \ex field when it is expecting to find a \exFr field. It assigns the existing \enEn field under the current \ex field, hypothesizes a missing \exFr sub-field, and then begins with the second found \ex field. By way of an error message, it prints out an error message in the hypothesized \exFr field:

```
\ex    Yax bo'on ta sna Antonio.
\exEn I'm going to Antonio's house.|
\exFr  ***Missing field inserted***
\ex    Ban yax ba'at?
\exEn Where are you going?
\exFr  Ou allez-vous?
```

Later, the user can search for the error strings (by default these are flanked by '***') and make the appropriate repairs.

In general, when the parser encounters an unexpected field, it assumes that a single field is missing, and attempts to repair the error by inserting the expected field, then resuming the parse with the next actual field. The reasoning here is that fields are more often missing than inserted or put in the wrong order.

However, not all parsing errors can be repaired in this way. If an unexpected field is encountered which cannot be repaired by inserting a single missing field before it, then the unexpected field is labeled with an error message, and the parser attempts to resume with the next field marker, ignoring the presumably erroneous one. Consider the following record, which is ill-formed in the light of the earlier definition:

```
\w    yax
\pos  AUX-V
\pos  Adj
\defn green
```

Since within a record only one \pos field is expected (in the absence of a \num field indicating multiple senses), the parser labels the second \pos field as erroneous, and attempts to resume parsing with the \defn field:

```
\w    yax
\pos  AUX-V
\pos  Adj  ***Erroneous field***
\defn green
```

If neither repair—insertion of a single field, or over-looking a single field—succeeds, then the parser issues a general error message "***Unable to parse record structure***", and resumes parsing with the next record.

Obviously this simple-minded error correction algorithm can go astray, but it flags many errors correctly, and when it cannot determine the cause of an error, it will at least tell the user that there is a problem in the record structure.

An alternative to using a special purpose parsing algorithm would be to export the dictionary as an XML file from Shoebox, and to use existing XML parsing tools. However, while Shoebox can export an XML file, it cannot import one. This approach would therefore require a separate XML lexicon viewer, with many of the capabilities of Shoebox built in; the user would have to locate an error in the XML viewer, then search in Shoebox for the same record in order to repair the error. By instead parsing the SMF-coded file directly and writing the error messages into the SFM file, the errors can be displayed directly in Shoebox.

Occurrence Counts

Shoebox can restrict the contents of designated fields to a certain set of elements, termed a "Range Set." This is useful for closed class items, such as parts of speech. The list of allowable elements can either be built by hand, or Shoebox will build it automatically from the actual elements found in the data. In my experience, if field linguists employ range sets at all, the latter is the way the sets are built—which means that any erroneous items in the data are automatically added to the range set.

A savvy user can examine the range set and remove any spurious items, then run a consistency check to repair any fields which violate the edited range set. But in fact, it often devolves upon a consultant to perform this check (if not to perform the repairs). While obvious errors are easy to spot (the use of both "Noun" and "noun", say), the consultant may not be familiar enough with the grammar of the language to notice other erroneous items in the range set. For this reason, it is useful to count the number of times particular elements in a given field appear, on the principle that what is rare is often an error.

There are many ways this can be done; I use a simple program (coded in Python) which counts all the strings appearing between a particular pair of regular expressions. For counting parts of speech, for example, the search expression has "^\\pos " (a "\pos" at the beginning of line)

to the left, and "$" (end of line) to the right. The resulting list can be perused for low-frequency items.

Spell Correction

Spell checking can easily be done for most major languages by extracting the text from fields which are in the desired language, and running the extracted text through an off-line spell checker (such as *aspell* or *ispell*[7]).

One problem with this approach is that SFM-coded fields may not be contained on a single line. This is particularly true of example sentences (or their translations into the glossing language(s)). It is therefore not sufficient to *grep* out the lines containing the desired SFM codes, without first normalizing the file(s) so that each field occupies a single line. Again, this can be done in a variety of ways; I use a simple Python program to combine all the fields of a given record onto a single line, then break the record up into fields again at SFMs. (I also tokenize the result into words, and sort them uniquely so that each word need only be checked once; abbreviations and the SFMs themselves can also be filtered out at this stage.)

Another detail that could cause problems is the encoding issue. Spell checkers assume a particular encoding, and if the Shoebox dictionary uses a different encoding, it would be necessary to run the text through an encoding converter (such as *iconv*[8]) prior to spell checking.

However, the biggest issue for spell *checking* of a multilingual dictionary is that it is cumbersome to do spell *correction*. That is, while *aspell* supports spelling correction of a monolingual file, it is not easy to merge the corrected result back into the SFM-encoded dictionary, even if one does not tokenize the extracted fields. Nor would it be straightforward to run *aspell* directly on the SFM-encoded files, precisely because they are multilingual, and there is no way to tell *aspell* what language a given field is in.

If there are only a small number of spelling errors, this is perhaps not an issue. One can extract the fields, run them through a spell checker to produce a list of misspelled words, then use Shoebox to search for each of the misspellings in situ.

But a dictionary I was recently working with prompted me to find another solution: the glosses were in both English and French, and the French glosses had been entered without accents or cedillas. Spell correction was therefore a massive exercise, involving not only correction of typos, but entering numerous accented characters.

The better solution involved exporting the SFM dictionary to Microsoft Word, running a program in Word to define the language for each field, and using Word's built-in French and English spell correctors on their respective fields. The French spell corrector made it trivial to add the accented characters. (Of course Word could not automatically correct words where two forms existed which differed only by the presence of accents: *a* 'has' and *à* 'to', for instance.) The file was then exported back into Shoebox.

Note that this process uses Word only as a temporary way of modifying the dictionary. It is not intended that any sort of editing, apart from spell correction, be performed in Word, thus avoiding the problems inherent in doing lexicography in a word processor (Bird and Simons 2003).

In more detail, the SFM language marking program is written in Word's Visual Basic programming language, and functions in effect as a Word macro. The user imports an SFM-coded file into Word, then launches the program from within Word.

The SFM language marking program parses the information on fields and the language that they are encoded in from the Dictionary Type file (see footnote 4), making certain assumptions. For example, Word has separate spelling dictionaries for several dialects of French; if the user specifies "French" in the type file, the import program assumes this means what Word calls "French (France)".[9] The SFM language marking program then automatically assigns the contents of each field in the SFM-coded file to the appropriate language. If a field uses the "Default" language, the program marks the field as not to be spell-checked. (The SFMs themselves are also marked not to be spell-checked.)

Once the program has assigned the field contents to the appropriate languages, the user can use Word's spell checking/ correction features to correct the spelling. When finished, the user saves the file as text, allowing it to be imported back into Shoebox.

Finally, not all languages of interest have spell checkers or correctors. In particular, it is unlikely that the target language of a minority language dictionary will have any spell checking facilities (and building an *aspell* dictionary from the contents of a bilingual dictionary is obviously not an option, since it is the bilingual dictionary itself that is to be checked!). However, what can be done is to extract the relevant fields (as described above for *aspell*), and feed them into a character n-gram program to produce lists of n-grams of various lengths. Token counts on the various n-grams can then be used to find rare n-grams, which may be errors. Another approach would be to parse the input into syllables, although I have not tried this as yet.

Reciprocal Cross-references

Shoebox has the built-in capability of checking that for every cross-reference, the target of that cross-reference exists. However, Chris Manning (p.c.) has suggested that one should also check for bidirectional references (e.g. synonyms), and this checking capability is not built into Shoebox. This sort of check is easily done by the following method.

[7] Both *aspell* and the similar *ispell* program are freely available, and run under Linux or the CygWin environment under Windows, as well as coming in native Windows versions. There are dozens of language-particular dictionaries for *aspell* and *ispell*, see http://aspell.net/ and http://fmg-www.cs.ucla.edu/geoff/ispell-dictionaries.html.

[8] Again, *iconv* is freely available.

[9] A list of installed languages is available from Word's Language dialog box.

Create a projection of the lexicon containing the cross-reference field and the field it cross-references. For example, if the cross-reference field is \syn, and this is intended to point to the \w (headword) field, the projection would consist of records containing the \w and \syn fields of all records containing a \syn field. The records of the projection are then formatted in a file so that the fields are on the same line (for convenience, a tab character can be used to separate the fields). A copy of this file is then made, with the fields in the opposite order. Both files are sorted, and then diff'd. Any lines appearing in one file but not the other represent one-direction cross references.

References

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