

HORN MORPHO 1.0 User's Guide

Michael Gasser

Indiana University, School of Informatics and Computing

`gasser@cs.indiana.edu`

7 December, 2009

1. Introduction

HORN MORPHO is a Python program that analyzes Amharic and Tigrinya words into their constituent morphemes (meaningful parts) and generates words, given a root and a representation of the word's grammatical structure. It is part of the L³ project at Indiana University <<http://www.cs.indiana.edu/~gasser/Research/projects.html>>, which is dedicated to developing computational tools for under-resourced languages. Later we plan to expand the program to handle other languages spoken in the Horn of Africa.

Natural language applications, such as question-answering, speech recognition, information retrieval, and machine translation, rely on a lexicon of possible forms in the relevant language. Morphological analysis is important for morphologically complex languages like Amharic and Tigrinya because it is practically impossible to store all possible words in a lexicon, and many words have close to 0 probability of occurrence in any given corpus. This becomes obvious in the context of machine translation to a morphologically simple language such as English, where the correspondence between words in Amharic or Tigrinya and the other language will often be many-to-one. The Amharic word ብይክረት ላቸውም, for example (which incidentally does occur in an online corpus), could be translated as 'even if it isn't opened for them'. While a system for processing English could include all of the English words in the translation (*even, if, it, isn't, opened, for, them*) in its lexicon, an Amharic system that includes all words such as ብይክረት ላቸውም is clearly impractical. For translation into Amharic or Tigrinya and sophisticated question-answering, morphological generation is also desirable because it is probably impossible to store all of the words that the system will output.

HORN MORPHO requires Python 2.5 or 2.6; it will not run under Python 3.¹ In order to use the program, you must also have a Unicode Ge'ez (Ethiopic) font such as Ethiopia Jiret or GF Zemen² on your computer. In addition, if you want to use the program to analyze particular words, rather than (or in addition to) Amharic or Tigrinya text in a file, you will need a way to enter Ge'ez characters. HORN MORPHO has been tested under Windows, Linux, and MacOS. Some of the options are not possible with Windows because of issues related to the display of non-roman characters (see the

¹ To download Python to your computer, go to <<http://www.python.org/download/>>

² Downloadable at <<http://www.senamirmir.com/download/jiret.zip>> and <<ftp://ftp.ethiopic.org/pub/fonts/TrueType/gfzemen.ttf>> respectively.

discussion in Section 8.b), but the main functionality of the program applies across all three platforms.

HORN MORPHO is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. See the GNU General Public License for more details. You should have received a copy of the license along with HORN MORPHO. If not, see <http://www.gnu.org/licenses/>.

HORN MORPHO is a work in progress, and it is certain to have bugs, as well as the limitations described in the last section of this document and features that make it awkward to use. Questions, comments, corrections, and suggestions (to gasser@cs.indiana.edu) are very welcome.

In the rest of this document, it is assumed that you have at least a basic knowledge of Amharic and/or Tigrinya and basic grammatical terminology. No familiarity with Python is assumed.

2. Installation

If you haven't already done it, uncompress the file that you downloaded. This will yield a directory (folder) called `HornMorpho-1.0`, which contains all of the files that you need to run HORN MORPHO.

The program is most useful if you install it on your computer; this places the program files in the place where other third-party Python programs reside on your system so that you can use them wherever you are running Python. If you are on a system with multiple users and separate administrative privileges, you will need these privileges to install the program, or you can ask a system administrator to do it for you.

To install HORN MORPHO, you will need to open a shell (often called “command prompt window” in Windows).³ In the shell, go to the `HornMorpho-1.0` directory (folder), and enter the following if you are on a Unix or Unix-like system:

```
python setup.py install
```

If you are using Windows, it will probably suffice to enter:

```
setup.py install
```

To test whether the installation succeeded, start up the Python interpreter (see Section 8.a if you don't know how to do this), and type

```
import l3
```

If you don't want to install the program, you can still use it. Just move the whole directory to a convenient place in your file system, and then make sure you run the Python interpreter from the `HornMorpho-1.0` directory, wherever that is. See Section 8.a for details.

³ How you open a shell depends on your operating system. In Unix or Unix-like systems, including MacOS, you run a terminal application (in MacOS this is found in Applications/Utilities). On computers running Windows, you need to run “Command Prompt” in Vista, “cmd.exe” in other versions.

The next section describes the conventions used in this document and in HORN MORPHO for representing Amharic and Tigrinya words using roman characters. Section 4 discusses the theory behind the program. If you just want to use the program and are not interested in the theoretical background, you can skip that section and go straight to Sections 5 and 6, which are concerned with aspects of Amharic and Tigrinya morphology and phonology that are implemented in the program. Section 7 deals with implementation details, in particular, the format of the data files; you can skip this section if you don't intend to look at any of these files. Section 8 is the most important section for most users; it describes the functions available in the program. Section 9 lists some of the limitations of the current version of the program.

3. Conventions

In the rest of the document, Amharic and Tigrinya words and morphemes will sometimes be written in Ge'ez characters and sometimes romanized. The romanization follows the conventions of the SERA system (Yitna and Yacob, 1996), with the following exceptions.

1. **Gemination** (consonant lengthening; ጥብቀት) is not normally indicated in the Ge'ez script. However, since it plays a significant role in the morphology of both languages and will be important (in later versions of HORN MORPHO) for speech applications, it is shown in the romanized forms of words, both in this document and within HORN MORPHO. The underscore character following the consonant indicates gemination: *teTeq_emeb_et* (ተጥቀመበት).
2. Ge'ez script does not distinguish between consonants that are not followed by a vowel and consonants that are followed by the high central vowel (phonetically [ɨ], sometimes also represented as [ə]) that the language uses to break up clusters of consonants, known in linguistics as an “epenthetic” vowel (Amharic ሰርጎ-ገብ, Baye, 2000 E.C.). Both are represented by the sixth order (ሳድስ) character in a series. Again because this distinction matters for speech applications, it is usually made here, with the character *I* used for this vowel: *zIfIS_Im* (ዝፍጽም).
3. SERA uses two-character sequences to represent three consonants: `s (**ሠ**, etc.), `h (**ሀ**, etc.), `S (**ሐ**, etc.). However, the backquote is also used for the consonant in the **o** series in Tigrinya. To avoid confusion between sequences such as ሰስ and ሠ, the backquote is here replaced by the caret (^), for example, in *^ser_a* (**ሠራ**).
4. All bare occurrences of vowels in Amharic are preceded by either an apostrophe (representing the **k** series) or a backquote (representing the **o** series). This is a more direct representation of Ge'ez orthography and agrees with Tigrinya, where these two series always represent consonants plus vowels. For example, *'IbEtu 'al_e* (**እቤቱ አለ**).
5. Tigrinya has a vowel, phonetically [ɛ] or [æ], in addition to those represented by the seven Ge'ez orders, that is of somewhat uncertain status. It appears after the laryngeal consonants ' , ` , *h*, *H*, where we would otherwise expect the vowel *e*, and is sometimes written with the first order (ግዕዝ) character, sometimes with the fifth order (አምስ) character, for example, በጽሐ / በጽሐ, መጺአን / መጺኤን. In HORN MORPHO, this Tigrinya vowel is represented by @: *beSH@*, *meSi'@n*.

Other than the cases noted in 3 and 4, HORN MORPHO uses the SERA conventions for Amharic and Tigrinya consonants: *ᵛ ha, ᵗ le, ᵗ Ha, ᵛ me, ᵛ ^se, ᵗ re, ᵗ se, ᵗ xe, ᵗ qe, ᵗ Qe, ᵗ be, ᵗ te, ᵗ ce, ᵗ ^he, ᵗ ne, ᵗ Ne, ᵗ 'a, ᵗ ke, ᵗ Ke, ᵛ we, ᵛ `a, ᵗ ze, ᵗ Ze, ᵗ ye, ᵗ de, ᵗ je, ᵗ ge, ᵗ Te, ᵗ Ce, ᵗ Pe, ᵗ Se, ᵗ ^Se, ᵗ fe, ᵗ pe*. Labialized consonants are represented with a following *W*, for example, *ᵗ qWe, ᵗ bWa*. The vowels, in traditional Ge'ez order are *e, u, i, a, E, (I), o*.

A few other conventions apply to the representation of verb roots; these are discussed below.

4. Theory

a. FINITE STATE MORPHOLOGY

When analyzing words, HORN MORPHO takes as input an Amharic or Tigrinya word in Ge'ez script. The word is first romanized, using the variant of the SERA romanization system described above. Next the program checks to determine whether the word is stored in a list of unanalyzable or pre-analyzed words. If not, it attempts to perform a morphological analysis of the word. It first does this using a “lexical” approach, based on a built-in lexicon of roots or stems and its knowledge of Amharic or Tigrinya morphology. If this fails, it tries to guess what the structure of the word is, using only its knowledge of morphology and the general structure of roots or stems. If the “guesser” analyzer fails, the program gives up.

Both the lexical and guesser analyzers operate within the general framework known as **finite state morphology**.⁴ Morphology (and phonology) is traditionally viewed within linguistics as the relationship between a **surface** form and a corresponding **lexical** (or underlying) form. For example, the surface Amharic form **ከቤታችን** can be analyzed into the sequence of three morphemes **ከ+ቤት+አችን** or, more abstractly, into a lexical representation that makes the structure of the word more explicit:

(1) [stem=**ቤት**, possessor=[person=1, number=plural], preposition=**ከ**]

In ordinary English, this says that the word **ከቤታችን** is derived from the noun stem **ቤት**, with a first person plural possessor (‘our’) and the preposition **ከ**. The knowledge that an Amharic noun stem can be preceded by a preposition and followed by a possessive suffix is part of Amharic **morphotactics**.

For example, one stage in the analysis of the word **ሰተና** *setena* ‘we drank’ is the conversion of the second vowel *e* into the sequence *ey*, revealing the third root consonant *y* of the verb whose complete root is *sty*. Expressed in romanized form:

(2) *setena* → *seteyna*

The arrow represents the direction of **analysis**, from a form that is closer to the surface to a form that is closer to the lexical representation. The reverse direction is that of **generation**, from a more abstract to a less abstract form (closer to the surface). Seen from the perspective of generation, (2) represents the merging of the *ey* sequence to the single vowel *e*. This is an example of a **phonological rule**: it combines two vowels to make the word conform to the constraints of Tigrinya phonology. It is also an **orthographic rule** because the resulting vowel also appears in the conventional

⁴ For a more in-depth introduction to finite state morphology, see Beesley & Karttunen (2003), and for another example of the application of finite state technology to Amharic morphology, see Saba & Girma (2006).

spelling of the word. Rules of both types are known within the finite state morphology community as **alternation rules** because they reflect alternations in the form of morphemes when they come together.

One complication that arises when we think of morphological analysis and generation in terms of a set of alternation rules is that the rules often have to apply in a particular order. For example, there is another rule which optionally converts the sequence *ey* to *E*, as in, **ዘይበልኩ** / **ዘበልኩ** *zeybelku* / *zEbelku*. Consider the order of these two rules in the generation direction, assuming that the input to the rules is the relatively abstract form *setey+na*. If the *ey*→*E* rule precedes the *ey*→*e* rule (as in 3a below), then we will get a form like *setEna* and will fail to get a form like *setena*. In the other direction (as in 3b), we get a form like *setena*, but nothing like *setEna* (which is apparently rare or non-occurring). Thus the order of the rules matters.

- (3a) 1. *ey*→*E* rule: *seteyna* → *setEna*
- 2. *ey*→*e* rule: *setEna* (rule fails to apply)
- (3b) 1. *ey*→*e* rule: *seteyna* → *setena*
- 2. *ey*→*E* rule: *setena* (rule fails to apply)

An FST consists of a set of **states**, joined by **transitions**, with one state designated the **initial state**. Each transition has a condition on it, consisting of an input and an output character, either of which may be zero (the empty character). To transduce an input string into an output string, we start in a start state and then, as long as it is possible, move from one state to the next by replacing an input character with an output character. That is, as the FST “consumes” the characters in the input string, one by one, it adds characters to a resulting output string. If we reach the end of the input string in one of the designated **final states** of the FST, the transduction has succeeded and the resulting output string is returned.

For example, consider a simple FST that is responsible for the merging of the sequence *ey* into the single vowel *e* in a word like **ሰተና** *setena* above, shown in Figure 1 below.⁵ In the figure, states are represented by circles and transitions by arrows. Input and output character pairs are separated by colons and multiple conditions on the same transition are separated by semicolons. A single character on a transition represents the same input and output character, and the absence of a character on either side of a colon represents an empty character. The character *C* represents any consonant, and *C* alone represents the same input and output consonant. *C*-*y* represents any consonant but *y*. The initial state is 0, and the circles with double borders (in this case, all states) represent final states.

Given the input string *seteye*, this FST outputs the same string because it goes eventually from state 0 to state 1 to state 2, then back to state 0, and none of the transitions changes the input character to a different output character. Given the input string *setena*, there are two possibilities. Either it outputs the same string, this time staying in states 0 and 1, or it outputs the string *seteyna*, passing through states 1 and 3 between the *t* and *n*, as shown in (4).

- (4) 0 *s:s* 0 *e:e* 1 *t:t* 0 *e:e* 1 *:y* 3 *n:n* 0 *a:a* 0

Note also that the FST fails on the input string *seteyna*; after state 2 is reached following the *ey*, there is no state that can be reached with an input *n*.

⁵ The rule shown in the figure is an oversimplification in several ways of what happens in the language and what is implemented in HORN MORPHO.

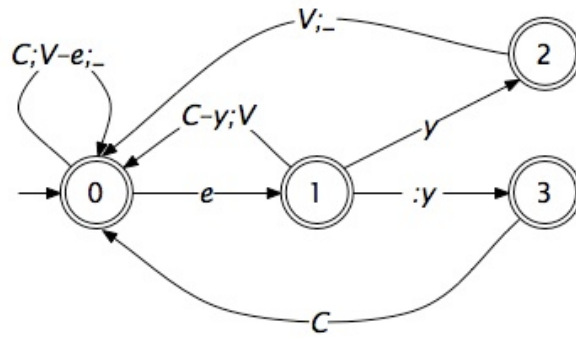


Figure 1. FST for converting surface *e* to lexical *ey* in Tigrinya.

Among the great achievements of the theorists of finite state morphology (Johnson, 1972; Kaplan & Kay, 1994; Karttunen et al., 1992; Koskenniemi, 1983) are the following results.

- All of the normal alternation rules known from natural languages, as well as morphotactics, can be captured by FSTs; that is, more complex devices requiring stacks, for example, are not required. Furthermore, since FSTs are invertible, an FST for analysis can be trivially converted into one for generation.
- A lexicon of roots or stems can be incorporated into a morphotactic FST.
- A series of FSTs, for example, implementing a sequence of ordered alternation rules and the morphotactics for a particular word, can be **composed** into a single FST which behaves exactly like the series of FSTs when applied to an input in the order that they are composed.

The upshot of these three facts is that it is possible to build a single FST that handles all aspects of the morphology for a particular class of words in a language, for example, an FST that analyzes all Amharic nouns.

A **deterministic** FST is one for which there is only one possible next state for a given unconsumed input string. For a given input string, a deterministic FST yields either nothing, if it fails on the string, or a single output string. In general, words in natural languages may be ambiguous; that is, a given word may have more than one legitimate analysis. Consider the Amharic orthographic word ብትሰማ. This has at least four possible analyses, corresponding to the English translations ‘if you hear’, ‘if you are heard’, ‘if she hears’, and ‘if she is heard’. For most applications, we would like a morphological analyzer to return all of the possible analyses. Therefore the FSTs that HORN MORPHO uses are **non-deterministic**; for a given input string and FST, there may be multiple successful paths through the FST, each corresponding to a different grammatical analysis of the string.

b. HANDLING NON-CONCATENATIVE MORPHOLOGY

Semitic languages like Amharic and Tigrinya, in particular their verbs, present a problem for finite state morphology, a problem that is well-known in the literature. Finite state morphology, without the introduction of special-purpose mechanisms, becomes very inefficient for **non-concatenative morphology**, that is, any case in which words don’t consist simply of sequences of morphemes or morphemes don’t consist simply of sequences of characters.⁶ Amharic and Tigrinya morphology is non-concatenative in at least two ways.

⁶ For a more complete discussion of non-concatenative finite state morphology, see Cohen-Sygal & Wintner (2006).

First, there are discontinuous morphemes (so-called circumfixes), which are separated by other morphemes, for example, the morpheme that indicates negation in non-subordinate Tigrinya verbs: **አይፈለጠን** *ay-feleTe-n*. A related problem is that the morphotactics of the end of the verb depends on what occurs at the beginning. For example, the Tigrinya relative prefix **ዝ** preceding the negative prefix normally precludes the negative suffix **ን**: **ዘይፈለጠ ነይሩ** *zI-ay-feleTe*.

Second, as in other Semitic languages, Amharic and Tigrinya verb stems have a so-called **root-template** structure. Consider these Amharic verbs, all based on the verb root meaning ‘repeat’:

- (5) **ይደግማል** *yIdegmal*
ይደግማል *yId_eg_emal*
ያስደግማል *yasdeg_Imal*
ይደጋግማል *yIdegag_Imal*

Each of these verbs has the prefix and suffix that indicate a verb in the imperfective tense/aspect (corresponding roughly to English present tense) with a third person singular masculine subject (‘he’ or ‘it’). What is left with the prefix and suffix removed is the verb **stem** (Amharic **አምድ**, Tigrinya **ግምድ**). Each of these stems is composed of two parts, the **root** (Amharic **ደግ**, Tigrinya **ደግ**), consisting of the consonant sequence *dgm*, and the **template**, consisting of a pattern of vowels and in some cases a prefix or the gemination of one or more consonants (the gemination is of course not indicated in Amharic orthography). For example, the template in the first case consists of the vowel *e* between the first and second root consonants and no vowel between the second and third consonants, and the template in the second case consists of the vowel *e* in both positions and the gemination of the first and second consonants. We can represent the templates using C_1 , etc. to represent the stem consonants.

- (6) $dgm + C_1eC_2C_3 \rightarrow degm$
 $dgm + C_1C_1eC_2C_2eC_3 \rightarrow d_eg_em$

Root-template morphology is non-concatenative because the two morphemes that make up the stem, that is, the root and the template, are interleaved with one another rather than being concatenated; as with the negative circumfix, the root and template are discontinuous.

Non-concatenative morphology presents a problem because finite state morphology has no memory other than the states themselves. For example, consider how an FST would handle a negative Tigrinya verb like **አይፈለጠን** *’ayfeleTen*. Once the initial *a* and *y* have been consumed by the FST, it needs to “know” that the verb is negative so that it will expect the *n* at the end. But how will it remember this while the stem of the verb (**ፈለጠ**) is being processed? The only option would seem to be an increase in the number of states, with separate negative and affirmative states for each of the positions between the negative prefix and suffix. But there are a number of other dependencies between prefixes and suffixes, and the required increase in the number of states would be sizable.

To avoid the massive duplication that this approach would require, finite state morphology researchers have introduced several techniques to provide limited memory or divide the work of recognizing different morphemes among separate parallel FSTs (see Cohen-Sygal and Wintner, 2006 for an extended discussion). HORN MORPHO makes use of the proposal of Amtrup (2003), which adds the possibility of grammatical constraints on the FST transitions in addition to the input-output character conditions. The application of this proposal to the analysis and generation of Tigrinya verbs is described at length in Gasser (2009). Its application to Amharic and Tigrinya is summarized in what follows.

One way of representing grammatical structure is through so-called **feature structures (FSs)** (Carpenter, 1992), consisting of sets of feature-value pairs. An Amharic example is provided by (1) above, repeated here for convenience, with abbreviated feature and value names:

(7) [stem=ቤጎ, poss=[pers=1, num=plur], prep=ke]

The features in this FS include `stem`, `poss`, and `prep`; their values follows the equal signs. Note that the values of features can be simple strings, as in the case of `stem` and `prep` or FSs in their own right, as in the case of `poss`. The fundamental operation on FSs is **unification**, essentially a pattern-matching process. Informally, unification takes two FSs, and if they are compatible, it returns the combination of the two. The details of how unification works are complex, but a few examples should make the general notion clear. If we attempt to unify the FS in (7) with a second FS, [`conjunction=m`], the result is this more complex FS:

(8) [stem=ቤጎ, poss=[pers=1, num=plur], prep=ke, conj=m]

In fact this represents the structure of the grammatical Amharic word ከቤታችንም. If we attempt to unify the FS in (7) with the FS [`prep=be`], unification fails because the feature `prep` has two different values in the two FSs. Informally this corresponds to the constraint that a word can only have one preposition. Unification also fails if we attempt to unify the FS in (7) with the FS [`poss=[pers=1, num=sing]`] because the `num` feature in the FS that is the value of the `poss` feature has different values (`plur` and `sing`) in the two FSs. Informally, a noun cannot have both a first person plural possessor ('our') and a first person singular possessor ('my').

The insight of Amtrup's (2003) work was to recognize that sets of FSs could be added to the transitions in an FST, adding additional constraints to the possible paths through the FST, at the same time maintaining the fundamental properties of FSTs that make them desirable, in particular the possibility of concatenating, inverting, and composing FSTs. An FST augmented with FS constraints works as follows:

- Processing begins with not only an input string but also an initial FS set, representing any grammatical constraints that are given, and at any given point during processing, a current FS set is maintained along with the remaining unconsumed string.
- When there is an attempt to traverse a transition, as with ordinary FSTs, the input character on the transition must be present at the beginning of the unconsumed input string. In addition, the current FS set must unify with the FS set on the transition (if there is one). The result of unifying two FS sets is the set of all possible unifications of pairs from the two sets. If the set unification fails (no two pairs of FSs unify), it is impossible to traverse the transition. If the set unification succeeds, the resulting FS set (the set of all of the successful unifications of pairs from the two sets) becomes the new current FS set.
- For each successful path through the FST, the resulting output string *and* the resulting FS set are both returned. That is, each analysis consists of a string and set of possible grammatical representations.

FSTs augmented with FS constraints solve the problem of non-concatenative morphology because the FS set that accumulates along each possible path through the FST represents a sort of memory for what previous transitions have been traversed. Consider again the case of the Tigrinya negative circumfix. Figure 2 shows a simplified representation of an FST for handling negation in the perfective tense/aspect. The input character sequence <ay> is an abbreviation for a transition with a single

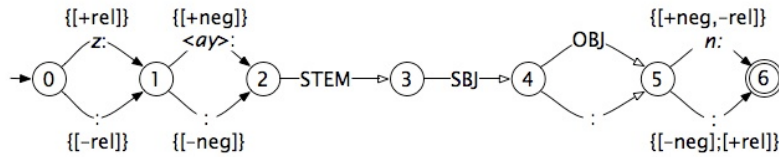


Figure 2. FST handling relativization and negation for perfective Tigrinya verbs.

character and one intervening state. The transitions with uppercase labels, STEM, SBJ, and OBJ, are abbreviations for the multiple states and transitions that are responsible for the stem, subject suffix, and optional object suffix of perfective verbs. Transitions labeled “:” consume no input character and add no output character. Six of the transitions have FS set constraints. The relevant features for this example are `rel`, which has the value `True` for relativized verbs such as **ዝደቀሰ** and **ዘይደቀሰ** and the feature `False` for non-relativized verbs such as **ደቀሰ** and **አይደቀሰን**, and `neg`, which has the value `True` for negative verbs such as **አይደቀሰን** and **ዘይደቀሰ** and the value `False` for affirmative verbs such as **ደቀሰ** and **ዝደቀሰ**. (The values `True` and `False` are abbreviated with + and – signs before the relevant features.) A semicolon separates different FSs within an FS set; the only example is on the lower transition from state 5 to state 6. The interpretation of this FS set is that this transition is compatible with either `-neg` or `+rel` (or both). The FST does not show how the prefix combination `z+ay` becomes `zey`; this results from a rule discussed in 6.a.

Now consider how this FST succeeds or fails on different input strings. In each case we assume that there are no initial grammatical constraints; that is, the initial FS set contains the single “top” element which unifies trivially with any FS. Presented with the string **ደቀሰ deq_ese**, Table 1 shows what happens. It is assumed that the STEM portion of the FST outputs the three-character root of the verb (the gemination character “_” is also part of the root) and that the SBJ portion of the FST accumulates the grammatical properties of the subject of the verb on the FS set.

Tables 2, 3 and 4 show the processing of the strings **አይደቀሰን aydeq_esen**, **ዘይደቀሰ zeydeq_ese** (*zeydeq_ese*), and the ungrammatical string **ዘይደቀሰን zeydeq_esen**.⁷ The analysis of *zeydeq_esen* fails because the final state in the FST is reached while the unconsumed input string is not empty (it still consists of the character *n*).

For other examples of the use of FS sets as grammatical constraints on transitions, see the discussion of Amharic morphology below and the examples of particular FSTs among the program data files.

5. Grammar

Any computational morphology system is based on a theory (explicit or implicit) of the grammar of the language. HORN MORPHO relies on a long history of research on Amharic and Tigrinya grammar, while introducing a few new notions. I have also been helped greatly by the output of Biniam Gebremichael’s web crawler for the languages, available at <http://www.cs.ru.nl/~biniam/geez/crawl.php>, which provides counts for 227,984 Tigrinya and 397,352 Amharic words found on the Internet. These data are extremely useful as a means of assessing current usage, especially for Tigrinya.

⁷ In fact **ዘይደቀሰን zeydeq_esen** is grammatical, but the *n* suffix at the end is the one that translates as ‘and’ rather than the negative suffix. This distinction is not handled by the simplified FST in Figure 2.

state	input string	output string	FS set
0	<i>deq_ese</i>		{}
1	<i>deq_ese</i>		{[-rel]}
2	<i>deq_ese</i>		{[-rel,-neg]}
3	<i>e</i>	<i>dq_s</i>	{[-rel,-neg]}
4		<i>dq_s</i>	{[-rel,-neg,subj=[...]]}
5		<i>dq_s</i>	{[-rel,-neg,subj=[...]]}
6		<i>dq_s</i>	{[-rel,-neg,subj=[...]]}

Table 1. Analysis of the string ደቀሰ deq_ese.

state	input string	output string	FS set
0	<i>aydeq_esen</i>		{}
1	<i>aydeq_esen</i>		{[-rel]}
2	<i>deq_esen</i>		{[-rel,+neg]}
3	<i>en</i>	<i>dq_s</i>	{[-rel,+neg]}
4	<i>n</i>	<i>dq_s</i>	{[-rel,+neg,subj=[...]]}
5	<i>n</i>	<i>dq_s</i>	{[-rel,+neg,subj=[...]]}
6		<i>dq_s</i>	{[-rel,+neg,subj=[...]]}

Table 2. Analysis of the string አይደቀሰን aydeq_esen.

state	input string	output string	FS set
0	<i>zaydeq_ese</i>		{}
1	<i>aydeq_ese</i>		{[+rel]}
2	<i>deq_ese</i>		{[+rel,+neg]}
3	<i>e</i>	<i>dq_s</i>	{[+rel,+neg]}
4		<i>dq_s</i>	{[+rel,+neg,subj=[...]]}
5		<i>dq_s</i>	{[+rel,+neg,subj=[...]]}
6		<i>dq_s</i>	{[+rel,+neg,subj=[...]]}

Table 3. Analysis of the string ዘይደቀሰ zeydeq_ese (zaydeq_ese).

This section is not meant to be anything like a thorough overview of Amharic and Tigrinya morphology; for more in-depth coverage, see Baye (2000 E.C.), Bender & Hailu (1978), or Leslau (1995) for Amharic; Leslau (1941) or Amanuel (1993) for Tigrinya.

An Amharic or Tigrinya word consists of a **lexical** part, or stem (Amharic አምድ, Tigrinya ኅምዲ), and one or more **grammatical** parts. This is easy to see with a noun, for example, the Amharic noun አገሮቻቸውንስ. The lexical part is the stem አገር; this conveys most of the important content in the

haric and 24 templates for Tigrinya, though it is rare for a root to occur with all. Different templates can be described as varying along three grammatical dimensions:

- **tense/aspect/mood**

Tense/aspect/mood (TAM) is signalled mainly by the prefixes and suffixes that indicate the subject of the verb but is also reflected in the stem template. In both languages, TAM has four possible values, traditionally referred to as **perfective** (or perfect), **imperfective** (or imperfect), **jussive/imperative**, and **gerundive** (or gerund). Here are examples from both languages of each with everything else held constant; in the romanization, the stem appears in boldface.

- perfective: ደረሰ *der_es-e* ገደፈ. *gedef-e*
- imperfective: ይደርሳል *yI-ders-al* ይገድፍ *yI-ged_If*
- jussive: ይድረስ *yI-dres* ይግደፍ *yI-gdef*
- imperative: ድረስ *dres* ግደፍ *gIdef*
- gerundive: ደርሶ *ders-o* ገዲፉ *gedif-u*

- **voice**

Voice has three possible values in Tigrinya, four in Amharic. It is signaled by the stem prefixes *te-* and *a-* for both languages and *as-* for Amharic, as well as particular patterns of vowels between the root consonants and gemination of the root consonants. I will refer to the plain form that has no prefix as **simplex** voice. Both languages have a further form that is conventionally called **passive/reflexive** (for simplicity, usually referred here to as “passive”). Amharic has two further forms, which I will call **transitive** and **causative**. In Tigrinya (as in Ge’ez) these correspond to a single form that I will call **transitive/causative** (for simplicity, usually shortened to “transitive”). These terms are only suggestive of the semantics, which in many cases depends on the particular root (for example, Amharic ተቀበለ, Tigrinya ተቐበለ is passive/reflexive in form but neither passive nor reflexive semantically). In fact, most Amharic roots fail to appear in all four forms, and many Tigrinya roots fail to appear in all three forms. Here are examples in both languages of each option in perfective and imperfective.

- simplex: ደረሰ *der_es-e* ይደርሳል *yI-ders-al*
- ገደፈ. *gedef-e* ይገድፍ *yI-ged_If*
- passive: ተደረሰ *teder_es-e* ይደረሳል *yI-d_er_es-al*
- ተገድፈ. *tegedf-e* ይግደፍ *yI-gId_ef*
- transitive: አደረሰ *ader_es-e* ያደርሳል *y-aders-al*
- transitive/causative: አግደፈ. *agdef-e* የግደፍ *y-agId_If* (*yegId_If*)
- causative: አስደረሰ *asder_es-e* ያስደርሳል *y-asder_Is-al*

- **stem-internal aspect**

In Amharic and Tigrinya, like other Semitic languages spoken in the Horn of Africa, there are two ways to modify stems through the introduction of the vowel *a*. In combination with different values for the voice feature, these two forms convey a wide range of subtle meanings, depending to a large extent on the particular verb root. I will refer to these two options by names which are based on their frequent meanings but by no means do justice to all of the possible interpretations. **Reciprocal** is formed by the insertion of *a* between the third and second consonants from the end of

the root; this can only occur in combination with passive and transitive voice. **Iterative** is formed by the reduplication (copying) of the second consonant from the end of the root and the insertion of *a* between the two copies of this consonant; this occurs in combination with all four voice options in Amharic (though only rarely with the causative) and all three voice options in Tigrinya. Other details of the templates depend on the values of the TAM and voice features. The plain form, with no *a* inserted in the stem, is again referred to as **simplex**. Here are examples of the three stem-internal aspect options, in combination with perfective TAM and both simplex and passive voice.

- Simplex aspect:

+simplex voice: ደረሰ <i>der_es-e</i>	+passive voice: ተደረሰ <i>teder_es-e</i>
ገደፈ <i>gedef-e</i>	ተገደፈ <i>tegedf-e</i>
- Reciprocal aspect:

+simplex voice: (not possible)	+passive voice: ተዳረሰ <i>tedar_es-e</i>
	ተጋደፈ <i>tegadef-e</i>
- Iterative aspect:

+simplex voice: ደራረሰ <i>derar_es-e</i>	+passive voice: ተደራረሰ <i>tederar_es-e</i>
ገዳደፈ <i>gedadef-e</i>	ተገዳደፈ <i>tegedadef-e</i>

ii. Stems: roots

The Amharic and Tigrinya verb stem is complicated not only because of the large number of combinations of TAM, voice, and stem-internal aspect; a whole set of additional complications results from the different categories that roots fall into. Like TAM, voice, and stem-internal aspect, the category of a root can affect the template vowels and consonant gemination.

HORN MORPHO relies on a new classification of root types, based on seminal earlier work by Bender & Hailu (1978) and also designed to reflect the commonalities between the Semitic languages, in particular for those cases which have only two overt root consonants in Amharic but three in Tigrinya and Tigre. Each root consists of a sequence of consonants and in addition, possibly a single *a*, a single character (“_”) indicating gemination, and a single character (“|”) following the first consonant indicating that is always “fused” to the next consonant. In the rest of this document, roots are written surrounded by angle brackets (“<>”). The root categories are described below. Each root example is followed by its **citation form**, that is, the third person singular masculine perfective form in simplex voice if this is possible for the root, otherwise passive or transitive voice.

• CCC(C)(C)

Most roots consist only of consonants. There are three main subcategories, differing in the number of consonants.

- CCC: simple three-consonant roots. Examples: <*sbr*> ሰበረ, <*mrT*> መረጠ; <*mrS*> መረጸ.
- CCCC: simple four-consonant roots. Examples: <*klkl*> ከለከለ, <*dngT*> ደነገጠ; <*dngS*> ደነገጸ.
- CCCCC: simple five-consonant roots (very rare). Examples: <*wxngr*> ወሸነገረ; <*nblbl*> ነበለለ.

• CC_C

Three-consonant roots for which the second consonant is geminated in most templates. In Am-

haric this category is indistinguishable from CCC in the perfective, distinguished in pronunciation but not writing in the imperfective and gerundive, and distinguished in both writing and pronunciation in the jussive/imperative: ይሰበር (CCC), ይጀምር (CC_C). In Tigrinya it is distinguishable in pronunciation but not writing from CCC in the perfective and gerundive, distinguished in both writing and pronunciation in the imperfective and jussive/imperative: ይሰብር, ይሰበር (CCC); ይጀምር, ይጀምር (CC_C). Examples: <Ty_q> ጠየቀ, <Cr_s> ጨረሰ; <Ty_q> ጠየቆ, <ws_k> ወሰኸ.

- **CaCC, CCaCC, C|CCaCC**

Roots in which the vowel *a* always appears before the second-to-last consonant. Examples: <gabz> ጋበዘ, <mark> ማረከ, <glamT> ገላመጠ; <bakn> ባኸነ, <mark> ማረኸ, <brabr> ተበራበረ; <bakn> ባኸነ, <mark> ማረኸ.

- **C|CaCC, C|CCCC, C|CCaCC**

Roots in which the first consonant is never followed by a vowel.⁹ These roots appear only in passive and transitive voice (never in simplex or causative). Examples: <n|saff> ተንሳፈፈ, <s|gbgb> ተስገበገበ, <n|xratt> ተንሸራተተ; <H|xkwxkxW> አሕሽኹሽኹ, <n|qsags> ተንቀሳቆሰ.

Particular root consonants result in templates that deviate from what is expected. This is especially true for Amharic.

- When any of the consonants in an Amharic root is ' or ` , and when the second root consonant is ungeminated *w* or *y*, the consonant is realized as a vowel; hence these roots are usually described as biconsonantal. Examples: <'sr> አሰረ, <s'm> ሰማ, <bl'> በላ, <'s_b> አሰበ, <TT_ '> ጠጣ, <qwm> ቆመ, <mwt> ሞተ, <xyT> ሸጠ, <hyd> ሄደ, <'nTs> አነጠሰ, <fnd'> ፈነዳ, <b'b'> ባባ.
- When any of the consonants in a Tigrinya root is a laryngeal (' , ` , *h*, or *H*), the vowel before or after this consonant may differ from what is usual: <'tw> አተወ, <SHf> ጸሓፈ, <bl'> በለፀ, <s`s''> ሳዕሰፀ.
- A special subcategory of Amharic CCC and CCCC roots consists of roots that originally ended in *y* and behave to some extent like CC' or CCC' roots. The symbol “*” is used to represent the final root consonant. Examples are <mx*> መሸ, <ff*> ፈጀ, <slc*> ሰለቸ, and <zgy*> ዘገየ. The exceptional roots <qr*> ቀረ and <sT*> ሰጠ, though they apparently never contained a *y*, are included in this subcategory because they behave like the other members.
- When the second or third consonant in a Tigrinya root is ungeminated *w* or *y*, the consonant is realized as a vowel in some templates. Examples: <xyT> ሸጠ, ይሸይጥ; <ftw> ፈተወ, ፈቶኸ, ይፈቱ; <sty> ሰተየ, ሰተኸ, ይሰቱ.
- In any of the four categories, root consonants may be **labialized** (Amharic የከኖፍር; Baye, 2000 E.C.), indicated in the romanization with a following *W*. In some templates, this may cause the following vowel to be *o* or *u*, sometimes optionally. In HORN MORPHO these changes are handled by an alternation rule. Examples: <kWr'> ኮራ (CCC), <dWldWm> ደለዶመ (CCCC), <bWacR>

⁹ Many of the C|CCCC roots are derived from CCC roots through the reduplication of the last two consonants, for example, Amharic *rgfgf* አርገፈገፈ from *rgf* ረገፈ. However, since this process is not particularly productive, and the meanings of the resulting C|CCCC roots are not always predictable, these are treated as independent roots in their own right in HORN MORPHO.

ቧጨረ (CaCC), <xmWaTT> አሽግጠጠ (C|CaCC); <HqWf> ሓጁፈ / ሓቆፈ (CCC), <mWg_s> ተጥገሰ (CC_C), <mrkWs> ተመርኩሰ / ተመርኮሰ (CCCC).

While most roots occur with all three values of stem-internal aspect, this is not the case for voice. A sizable subset of roots fail to occur in the simplex voice (in addition to the roots beginning with “C|”, which, as noted above, appear only in passive and transitive voice). Examples: <dr̥g> ተደረገ, <q̥b_l> ተቀበለ, <mlkt> ተመለከተ, <q̥m_T> ተቀመጠ, <ds_t> ተደሰተ; <zrb> ተዘርበ, ተዛረበ; <rd_’> ተረድኦ; <q̥b_l> ተቆበለ; <q̥m_T> ተቆመጠ; <Sb_y> ተጸበየ.

iii. Affixes

The verb in both languages has four slots for prefixes before the stem and four slots for suffixes after the stem. The positions of the affixes are as follows. Parentheses indicate optionality, and a vertical bar indicates different options within a given slot. Each of the affixes is described briefly below.

- (10) Amharic: (prep|conj)(rel)(neg) sbj
 STEM
 sbj (obj|def)(neg|aux|acc)(conj)
 Tigrinya: (prep|conj)(rel)(neg) sbj
 STEM
 sbj (obj) (neg) (conj)

An Amharic or Tigrinya verb must agree with its subject. Subject agreement (sbj) is expressed by suffixes in the perfective, gerundive, and imperative and by prefixes and suffixes (in the second person singular feminine and second and third person plural) in the imperfective and jussive. There are eight possible combinations of person, number, and gender in Amharic and ten possible combinations in Tigrinya. Some forms are ambiguous; for example, Amharic ይመጣሉ *yImeTal_u* can have a second or third person singular polite subject, as well as a third person plural subject; Tigrinya ትኸዱ *tIKedu* can have a second person masculine singular polite subject, as well as a second person masculine plural subject.

Following the subject agreement suffix, there may be an object suffix (obj). Here there are nine possible forms for Amharic because the second person polite form (-wo(t)) is distinguished from the third person plural (-ac_ew). Both languages also have the possibility of an indirect (or applicative or prepositional) object suffix. In Tigrinya, these are marked by the prefix -/ before the object suffix, with a range of meanings: ‘to, for, for the benefit of, on, at, from, to, with, by, to the detriment of’. In Amharic, indirect objects are marked by one of two prefixes before the object suffix: -/ - ‘to, for, for the benefit of’, -b_ - ‘on, at, from, to, with, by, to the detriment of’. Thus there are a total of 20 possible object suffixes in Tigrinya, 27 in Amharic. Examples of each type: አቀፈኛቸው *’aq_efec_-ac_ew* ‘she hugged **them**’, አቀፈኛላቸው *’aq_efec_-ll_ac_ew* ‘she hugged (somebody) **for them**’, አቀፈኛባቸው *’aq_efec_-lb_ac_ew* ‘she hugged (somebody) **to their detriment**’; ሓቆፋቶም *HaQwifat_om* ‘she hugged **them**’, ሓቆፋትሎም *HaQwifatll_om* ‘she hugged **for/against them**’.

Negation (neg) in both languages is indicated by a prefix and, for non-subordinate verbs (see below), a suffix as well: አልተረጋገጠም *’al-teregag_eTe-m*, አይተረጋገጽን *’ay-teregageSe-n* ‘it was **not** verified’; The gerundive is never negated.

In both languages, verbs in the perfective and imperfective may be **relativized** (rel) with the prefix *ye-* (perfective) or *yem_-/Im_-* (imperfective) for Amharic, *z-* for Tigrinya. A relativized verb fills

the verb position in a relative clause (Amharic አዛማጅ አረፍተ ነገር, Baye, 2000) and, as such, functions more like an adjective than a verb: የደረቀ እንጨት *ye-der_eqe* 'InCet, ዝደረቅ ዕንጨይቲ *zI-dereQe* 'InCeyti 'wood that dried; dry wood'. A relativized verb without a modified noun behaves like a noun clause: የደረቀው 'what dried; the dry thing'. Note that the third person singular masculine object suffix in Amharic (but not Tigrinya) doubles as a marker of **definiteness** (def) for relativized verbs: የደረቀው እንጨት *yeder_eqe-w* 'InCet 'the dry wood'.

Relativized verbs are **subordinate**; that is, they cannot act as the main verbs of sentences. Verbs can also be made subordinate through the addition of a prefix conjunctions (conj) such as Amharic *bI-* and *IndI-* and Tigrinya *nIKI-* and *'Inte-*. A verb with one of these prefixes functions as the verb in an adverbial clause: ሥጋ ብበላ *^sIga bI-bela* 'if I eat meat', በጊዜ እንዲደርስ *begizE 'Ind-iders* 'so that he arrives on time'; እንተረኝብካያ *'IntereKibkay_a* 'if you (masc. sing.) meet her', ንክንኸውን *nIK-In_IKew_In* 'so that we become'. Except for the negative following እንተ *'Inte-*, prefix conjunctions never co-occur with the relativizing prefix.

A noun that is modified by a relative clause may be the object of a preposition, and in Amharic and Tigrinya this preposition (prep) is prefixed to the relativized verb rather than the noun itself, as would be the case with an unmodified noun (see the section on nouns below). With a prepositional prefix, the Amharic relative prefix drops out in the perfective and is shortened to *-m_* in the imperfective. This does not happen in Tigrinya, and the preposition is often written as a separate word. Examples: ለወደቀው ተማሪ *le-wed_eqew temari* 'for the student who failed', በሚገኝበት ከተማ *be-m_ig_eN_Ib_et ketema* 'in the town where it is found'; ንዝሓለፉ 4 ዓመታት *nI-zIHalefu 4 `ametata* 'for the four years that passed'.

A noun that is modified by a relative clause may also be the direct object of a verb, and in Amharic it then requires the **accusative** (acc) suffix under some circumstances. As with prepositions, this suffix is added to the relativized verb rather than the noun: እጅ የሰጡትን ወታደሮች ማረኳቸው *'Ij_yeseT_ut-In wet_ad_eroc_mar_ekWac_ew* 'they captured the soldiers who surrendered'.

Amharic verbs in the imperfective and gerundive may take a form of the auxiliary (aux) verb አለ *'al_e* as a suffix. The auxiliary is required for affirmative, non-subordinate imperfective verbs in most contexts: ይወድቃሉ *yIwedq-al_u*. With the gerundive, the auxiliary creates a non-subordinate form similar to English present perfect: ወደቀዋል *wedqew-al* 'they have fallen'. The corresponding auxiliaries in Tigrinya are normally written as separate words, and they are not handled in HORN-MORPHO.

Finally, verbs in both languages may take one of a set of conjunctive or adverbial suffixes (conj): ትውጣና *tIwTa-n_a* 'let her go out and...', የተደበቀውም *yetedeb_eqew-m* 'also what was hidden'; ዘምጽኦምን *zemSI'om-In* 'who brought them and...', ተረዲኡኪዶ *tered_iuk_i-do* 'do you (fem. sing.) understand?'

b. NOUNS

An Amharic noun consists of a stem and zero or more prefixes and suffixes. Since Amharic nouns and adjectives are similar morphologically, they are not distinguished in HORN-MORPHO, and many of the words referred to as "nouns" in what follows are more properly thought of as adjectives.

i. Stems

Noun stems are of two types. Some are unanalyzable in that they are not derived from a simpler stem or root. Examples: ቤት *bEt*, ምሳ *mIsa*, ጨረቃ *Cereqa*.

Other stems are derived from verb, adjective, or noun roots. Like other Semitic languages, Amharic has many ways of deriving nouns from verbs. As with verb stems, each of these **deverbal noun** stems is formed through a combination of a verb root and a grammatical template. HORN MORPHO handles four categories of deverbal nouns, those that are possible with all verb roots.

- **infinitive**

The infinitive is recognizable by the stem prefix *m(e)-*. As with full-fledged verbs, an infinitive stem has values for voice and stem-internal aspect. The infinitive is the only deverbal noun that has a negative form. Examples: መቀረጥ *meqWreT*, root: <*qWrT*>, voice: simplex, aspect: simplex; አለማሳደብ *'alemas_adeb*, root: <*sdb*>, negative, voice: transitive, aspect: reciprocal.

- **instrumental**

The instrumental is based on the infinitive, differing only in the suffix and its possible effect on the last root consonant. The name “instrumental” comes from a common meaning of this form, the instrument that is used in the action associated with the verb, but it has other functions as well. Like the infinitive, it varies in terms of voice and stem-internal aspect. Examples: መክፈቻ *mek-feca*, root: <*kft*>, voice: simplex, aspect: simplex; መታጠቢያ *met_aTebiya*, root: <'Tb>, voice: passive, aspect: simplex.

- **agentive**

The agent stem may refer to the “doer” of the action of the verb, but it has other functions as well, including adjective-like functions. Like the infinitive, agentive forms vary in terms of voice and stem-internal aspect. Examples: ገዥ *geZ*, root: <*gz*>, voice: simplex, aspect: simplex; አወዳዳሪ *'aw_edadari*, root <*wdr*>, voice: transitive, aspect: iterative.

- **manner**

The manner stem refers to the manner or way in which the action of the verb is performed. Its template is based on the transitive iterative template, but it is invariant across all voice and aspect values. Examples: አነጋገር *'an_egager*, root: <*ngr*>, አስተዳደር *'astedader*, root: <'dr>.

Amharic noun stems are also derived from adjectives, for example, ነጻነት *neSa-nnet*, and nouns, for example, አብዮታዊ *'abiyot-awi*. Of these possibilities, only the *-awi* suffix is handled in HORN MORPHO; the program does not analyze stems such as ነጻነት further.

ii. Affixes

The Amharic noun has two prefix slots and four suffix slots. None of the affixes is obligatory. The possibilities are as follows. Each is described briefly below.

(11) (prep | gen) (distrib) STEM (plur) (poss | def) (acc) (conj)

The plural (plur) noun suffix is normally *-oc_*. Other cases are treated as irregular plurals; only some of these are known to HORN MORPHO, for example, መንግሥታት *mengI^st-at*.

There are nine possible possessive (poss) suffixes for the different combinations of person, number, and gender (including the second person polite suffix *-wo(t)*). The third person singular suffixes

are ambiguous since they double as definite (def) articles, for example, ክፍሉ *kɪfl-u* ‘his room; the room’.

If the noun is definite and the direct object of a verb, it must normally take the accusative (acc) suffix *-n*.

When a noun is the object of a preposition and not modified by an adjective or relative clause, the preposition (prep) is prefixed on the noun. The **genitive** (gen) marker *ye-* appears in the same position.

The distributive (distrib) prefix *Iy_e-* translates roughly as ‘each’.

Finally, more or less the same set of conjunctive or adverbial suffixes (conj) is possible with nouns as with verbs.

Some examples: ለየክልሉ *le-y_e-kll_II-u*, stem: *kll_II*, [prep=*le*, +distrib, +def]; የርምጃቸውንና *ye-rmlj_a-c_ew-n-In_a*, stem: *'IrmIj_a*, [+gen, +acc, poss= [pers=3, num=plur], conj=*na*]; ከወንድሞቻችሁም *ke-wendIm_-oc_-ac_Ihu-m*, stem: *wendIm_*, [prep=*ke*, +plur, [poss= [pers=2, num=plur], conj=*m*].

6. Alternation rules

As discussed in the last section, Amharic and Tigrinya are quite similar at the level of verb morphotactics, as well as with respect to the options that are available for each of the positions in the verb. But there is more to words than morphotactics, which is meant to apply as generally as possible. In specific cases, when particular phones appear in particular positions within words or particular combinations of phones come together, the surface wordform that results may be different from what the morphotactics would dictate. Alternation rules, implemented as finite state transducers, specify these changes. The alternation rules for Amharic and Tigrinya are very different, so they are dealt with separately in what follows.

a. ALLOMORPHY

i. Amharic

Allomorphy refers to variation in the form of a morpheme, depending on its context. HORN MORPHO includes three alternation rules for Amharic allomorphy, listed below along with the data files in which they are implemented and some examples.

1. Third person singular object suffix
t following *o* and *u*, *w* following other vowels, *ew* following consonants
Implemented in *3sm.fst*, where the suffix is represented on the lexical end by “3”
ወደዱት *wed_edu-t*, ወደደው *wed_ede-w*, ይወደዋል *ylwed_ew-al*
2. Consonant in the first person singular and second person masculine perfective subject suffixes
h following vowel, *h* or *k* following consonant
Implemented in *kh.fst*, where the consonant is represented on the lexical end by “7”
ረሳሁ *res_a-hu*, አልኩ *'al-ku*, አልሁ *'al-hu*

3. Third person singular possessive suffix, noun masculine definite suffix
u following consonants, *w* following vowels
 Implemented in `u2w.fst`.
ልጅ *llj-u*, **አልጋው** *'alga-w*.

ii. Tigrinya

HORN MORPHO includes two alternation rules for Tigrinya allomorphy.

1. Relativizing prefix
z-, optionally *I-* (geminating the next consonant) before a prefix beginning with *t* or *n*
 Implemented in `rel.fst`, where the prefix is represented as “R”
 Examples: **ዝሰበረ** *zI-sebere*, **ዝተሰበረ** *zI-tesebre*, **እተሰበረ** *'It-tesebre*
2. *a/e* in transitive and negative prefixes
 In the transitive stem prefix *a-* and the negative prefix *ay-*, the initial vowel is realized as *e* unless it starts the word (that is, after the consonant *'*).
 Implemented in `A.fst`, where the vowel is represented as “A”
 Examples: **አይረአየን** *'ayre'ayon*, **ዘይረአየ** *zeyre'ayo*, **አቐመጦም** *'aQem_eTom*, **ዘቐመጦም** *zeQem_eTom*

b. PHONOLOGY AND ORTHOGRAPHY

i. Amharic

When morphemes come in contact with one another in an Amharic word, there a number of general phonological processes and a few orthographic variants that apply, all of which can be captured in alternation rules. The following is a list of those that have been implemented in HORN MORPHO, along with the names of the files specifying the FSTs for the rules and one or more examples. Each of the rules is described as it applies in the generation direction, that is, from more abstract (lexical) to more concrete (surface). In the rule notation, “*C*” means any consonant (note that in the romanization of words it represents the consonant of **ጭ**); “*V*” means any vowel; “*#*” represents the end of a word; “*_*” indicates gemination of the preceding consonant; “[*]*” represents alternatives; other abbreviations are given following the rule. Unless otherwise indicated, the rules are obligatory.

1. $CC \rightarrow C_$
 Implemented in `gem.fst`
yIberral \rightarrow *yIber_al* (**ይበረል**)
2. $gk \mid qk \rightarrow k_$, $Tt \rightarrow t_$; optional
 Implemented `CC_ass.fst`
fel_egkut \rightarrow *fel_ek_ut* (**ፈለኩት**), *seTtoal* \rightarrow *set_oal* (\rightarrow *setW_al*) (**ሰቷል**)
3. $sS \rightarrow S_$; $S: \wedge s, S, \wedge S, x, z, Z$; only applies within verb stems (*as-* causative prefix)
 Implemented in `stem_ss.fst`
aszereg_a \rightarrow *az_ereg_a* (**አዘረጋ**)
4. $CWe \rightarrow Co$, $CW(I) \rightarrow Cu$; optional for *g, k, q*; only applies within verb stems
 Implemented in `stem_lab.fst`
xWel_eke \rightarrow *xol_eke* (**ሾለክ**)

5. $Ko \rightarrow KWe, Ku \rightarrow KW$; K : g, k, q ; optional; only applies within verb stems
Implemented in `stem_lab.fst`
 $qome \rightarrow qWeme$ (ቁመ)
6. $Cwa \mid Coa \mid Cua \rightarrow CWa$; optional
Implemented in `w2w.fst, ou2w.fst`
 $^sergwa \rightarrow ^sergWa$ (ሠርጻ), $sebslboal \rightarrow sebslBWal$ (ሰብስቧል)
7. $VE \rightarrow VyE, Vo \rightarrow Vwo, ia \rightarrow iya \mid Iya, ua \rightarrow uwa, oa \rightarrow owa$; mostly optional
Implemented in `n_c_epen.fst, y_epen.fst`
 $gelaE \rightarrow gelayE$ (ገላዩ), $gelaoc_ \rightarrow gelawoc_$ (ገላዎች), $tlfel_Igial_ex \rightarrow tlfel_Igiyal_ex$ (ትፈልጊያለኸ)
8. $aa \mid (e)ea \mid ae(a) \rightarrow a, eu \mid au \rightarrow u, ao \rightarrow o, ai \rightarrow i, ee \rightarrow e, VI \rightarrow V$; obligatory with exception of ao and ea in some circumstances
Implemented in `vv.fst, n_vv.fst, ao.fst, ea.fst`
 $glbau \rightarrow glbu$ (ግቡ), $qer_eec_ \rightarrow qer_ec_$ (ቀረች)
9. $IyC \rightarrow iC, IyV \rightarrow iyV$
Implemented in `y2i.fst`
 $siySIf \rightarrow siSIf$ (ሲጽፍ)
10. $JiV \mid J(_)EV \rightarrow J(_)V, Ji\# \rightarrow J\#$; J : c, C, j, N, x, y, Z ; the second rule is optional
Implemented in `vn_pal.fst`
 $teqem_IC_Eal_ehu \rightarrow teqem_IC_al_ehu$ (ተቀምጫለሁ)

Another phonological process characteristic of Amharic, **palatalization** (ላንቃዊነት in Baye, 2000 E.C), occurs only in certain morphological environments. Palatalization makes the following changes in consonants: $t \rightarrow c, d \rightarrow j, s \mid ^s \rightarrow x, z \rightarrow Z, T \mid S \mid ^S \rightarrow C, n \rightarrow N, l \rightarrow y$. It is implemented in the files `pal_iE.fst` (where “8” represents the palatalization context on the lexical side) and `vn_pal.fst`. It occurs before the vowels i and E in the following contexts.

1. Imperfective and imperative second person singular feminine subject suffix i
Example: $bert-i \rightarrow berci$ (በርቺ)
2. Gerundive first person singular subject suffix E (in this case the preceding consonant is always geminated)
Example: $keflI_E \rightarrow kefly_E$ (ክፍዩ)
3. Agentive suffix i (the i is usually dropped by rule 10 above if it ends the word)
Example: $Cek_an-i \rightarrow Cek_aNi$ ($\rightarrow Cek_aN$) (ጨካኝ)
4. Instrumental suffix ia (the i is later dropped by rule 10 above)
Example: $meCer_es-ia \rightarrow meCer_exia$ ($\rightarrow meCer_exa$) (መጨረሻ)

ii. Tigrinya

The following is a list of most of the Tigrinya phonological and orthographic rules that have been implemented in HORN MORPHO, along with the names of the files specifying the FSTs for the rules and one or more examples. This aspect of Tigrinya grammar is notoriously complex, and a number

of details, in particular concerning the interaction of the different rules, have been omitted. Each of the rules is described as it applies in the generation direction, that is, from more abstract (lexical) to more concrete (surface). In the rule notation, “*C*” means any consonant; “*V*” means any vowel; “#” indicates the end of a word; “\$” indicates a stem boundary; “_” indicates gemination of the preceding consonant; “|” represents alternatives. Unless otherwise indicated, the rules are obligatory. The first three rules apply only within verb stems; rule 4 applies at the boundary between stems and suffixes; rules 5, 6, and 7 apply within affixes; the other rules apply anywhere.

1. Within verb stems; $CWe \rightarrow Co$, $CW(I) \rightarrow Cu$; optional for *g*, *k*, *q*
Implemented in `stem_lab.fst`
***mWeleQe* → *moleQe* (ጠለቕ)**
2. Within verb stems; $Xo \rightarrow XWe$, $Xu \rightarrow XW$; *X*: *g*, *k*, *q*; optional
Implemented in `stem_lab.fst`
***qome* → *qWeme* (ቁመ)**
3. Within verb stems: $ew(e)C \rightarrow oC$, $ey(e)C \rightarrow eC$, $ewiC \rightarrow oyC$, $eyiC \rightarrow eyC$, $CweC \rightarrow CuC$, $CyeC \rightarrow CiC$
Implemented in `stem_wy.fst`
***mewete* → *mote* (ጠተ), *yIKeydu* → *yIKedu* (ይኸዱ), *kewinu* → *koynu* (ኮይኑ), *xeyiTu* → *xeyTu* (ኸይጡ), *qwem* → *qum* (ቁም), *kyedu* → *kidu* (ኪዱ)**
4. At stem-suffix boundaries: $ew\$C \rightarrow oC$, $ey\$C \rightarrow eC$, $ey\$e \rightarrow e$, $Iw\$ \{C\#\} \rightarrow u \{C\#\}$, $Iy\$ \{C\#\} \rightarrow i \{C\#\}$, $iw\$V | iy\$V \rightarrow yV$, $iw\$C | iy\$C \rightarrow iC$; vary in obligatoriness
Implemented in `vwy.fst`
***fetewka* → *fetoka* (→ *fetoKa*) (ፈቶካ), *deleyna* → *delena* (ደለና), *teSeb_eyen_i* → *teSeB_en_i* (ተጸበኒ), *yIfet_Iw* → *yIfet_u* (ይፈቱ), *`aSiwom* → *`aSyom* (ዓጽዮም), *qen_iykIn* → *qen_ikIn* (→ *qen_iKIn*) (ቀኒኸን)**
5. Within affixes: $Iy(I) \rightarrow i | I$, $Iye \rightarrow e$, $uw \rightarrow Iw$; the second rule optional
Implemented in `CyC.fst`, `u2Iw.fst`
***zIyIgeb_Ir* → *zigeB_Ir*, *zIgeb_Ir* (ዚገብር, ዝገብር), *'ayre'ayuwon* → *'ayre'ayIwon* (አይረአይዎን)**
6. Within affixes: $ea \rightarrow a$, $eo \rightarrow o$, $ee \rightarrow e$
Implemented in `VV.fst`
***re'ayeo* → *re'ayo* (ረአዮ), *feleTeen* → *feleTen* (ፈለጠን)**
7. Within affixes: $ey(yI) \rightarrow E$, $Iye \rightarrow E$; optional
Implemented in `ey2E.fst`
***zeytensI'u* → *zEtensI'u* (ይተንስአ), *zeyyISIg_Imek_a* → *zESIg_Imek_a* (ይጽግመካ)**
8. $CC \rightarrow C_$
Implemented in `gem.fst`
***yIseddu* → *yIsed_u* (ይስዱ)**
9. $Vk \rightarrow VK$, $Vq \rightarrow VQ$ (unless *k* or *q* is geminated)
Implemented in `KQ.fst`
***bekeye* → *beKeye* (በኸየ), *yIwIs_Ik* → *yIwIs_IK* (ይውስኸ), *ziqIm_eT* → *ziQIm_eT* (ዚቕመጥ)**

There is some variation in the vowels that occur before laryngeal consonants (‘, ` , *h*, or *H*) within verb stems. Some of this is handled in HORN MORPHO, but I am unclear on exactly what the rules should be. Examples: አይክአለን ‘ayke’alen, አይካአለን ‘ayka’alen, አይክአለን ‘aykI’alen.

7. Implementation

a. CASCADE OF COMPOSED FSTs

HORN MORPHO includes separate lexical and guesser analyzer FSTs for Amharic and Tigrinya verbs and for Amharic nouns, stored in the files Am/vb.fst, Am/vb0.fst, Ti/vb.fst, Ti/vb0.fst, Am/n.fst, and Am/n0.fst. There are also separate lexical and guesser generator FSTs, stored in Am/vbG.fst, Am/vb0G.fst, Ti/vbG.fst, Ti/vb0G.fst, Am/nG.fst, and Am/n0G.fst.

Though an analyzer FST can be trivially inverted to yield a generator FST, in the program, the generator FSTs are created separately because they do not need to handle the phonological or orthographic variation that is required by the analyzer FST. For example, the lexical analyzer for Amharic verbs yields a single analysis for the alternate spellings ሰርቆአል and ሰርቆል, but the generator that starts with this analysis produces only ሰርቆል.

There are also lexical analyzers for the **copula** (ነው, ናችሁ; እዩ, አና, etc.) in each language, treated separately because of its irregularities and stored in Am/cop.fst and Ti/cop.fst. For Amharic, the negative of the copula (አይደለም, አይደለንም, etc.) is also included in this FST; for Tigrinya, the negative of the copula (አይኮነን, etc.) is only recognized as the regular negative past perfective of the verb <kwn> ኮነ and not included in the copula FST. The past of the copula (ነበረ, ነበር, etc.) for both languages is treated only as the regular perfective of the verb <nbr>. The generators for the copula are created by inverting the analyzers; they are not stored separately in files.

Each of these 14 FSTs results from the composition of a **cascade** of simpler FSTs, each of which is responsible for an alternation rule, for the morphotactics of a word or stem belonging to a particular class, or for a list of lexical roots or stems. The overall structure of this cascade for the Amharic lexical verb FST is shown in Figure 3 and described in what follows. (The architecture for Tigrinya verbs is similar.) Each rectangle in the figure represents an FST; the symbol “o.” represents the composition operation (from lower to higher rectangle).

The files for the FSTs that are composed to make the complete FST are listed in the cascade file Am/vb.cas. By far the most important of these is the last one, the FST that covers the morphotactics of the verb, stored in the file Am/vb_max.fst and represented by the large yellow box in the figure. Within this there is a sequence of prefix and a sequence of suffix positions. These are concatenated onto the FST representing the verb stem (the darker yellow box). This is the output of the composition of an embedded cascade of FSTs, listed in the cascade file Am/vb_stem.cas. Two of the constituent FSTs (in red) are phonological, representing alternation rules that apply only to the verb stem. Three (in blue) are morphotactic, representing the structure of the stem, in particular how the root and template fit together, at three levels of abstraction. One (in purple) is the FST that encompasses all of the verb roots known to the system, stored in the lexical FST file Am/vb_root.lex. The FSTs making up the stem are composed in order from bottom (closer to the surface) to top (the lexical end). The verb stem FST can analyze a bare verb stem. For example,

$$(12) \text{ teTeyay}_q \rightarrow \text{Ty}_q; [\text{tm}=\text{ger}, \text{vc}=\text{ps}, \text{as}=\text{it}]$$

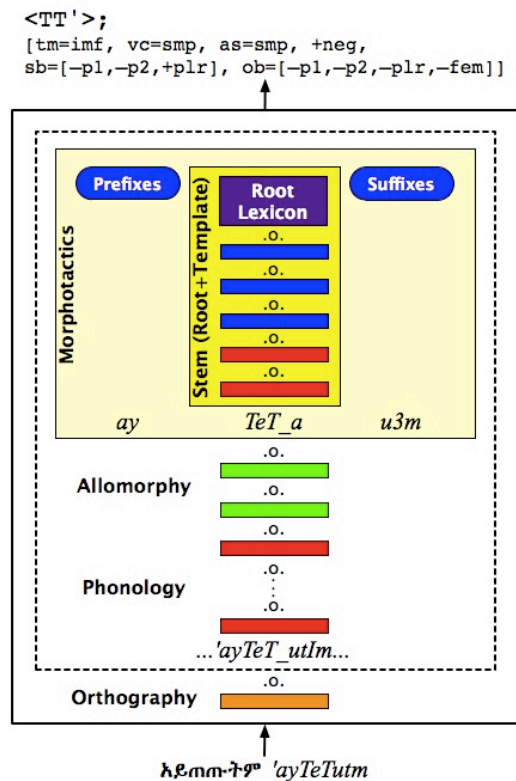


Figure 3. Cascade of FSTs in Amharic lexical verb analyzer

That is, the stem *teTeyay_q* has the root $\langle Ty_q \rangle$, gerundive TAM, passive voice, and iterative stem-internal aspect.

The remaining FSTs implement alternation rules that apply beyond the stem, two for the first two allomorphic rules described in Section 6.a. (in green in the figure), eleven for the phonological rules described in Section 6.b. and a few other miscellaneous rules (in red), and one to convert between the phonological representation of the input that includes gemination and the vowel *I* and a purely orthographic representation that omits these features (in orange).

The figure illustrates the analysis of the word አይጠጡትም ‘they don’t drink it’. This is first romanized to *’ayTeTutm*. The orthographic-to-phonological FST converts this to all possible pronunciations of the input string, including the correct one, shown in the figure above the dashed border that surrounds the phonological part of the system. This form and the other surviving strings are processed by the intervening phonological FSTs. One of these replaces *u* with *au*; another replaces *t* with *3*, the character representing the third person singular masculine object suffix. Among the strings that make it to the morphotactic FST is *ayTeT_au3m*, which is analyzable as the pair of prefixes *a+y*, the stem *TeT_a*, and the three suffixes *u+3+m*. The final output analysis is shown at the top of the figure. The root of the word is $\langle TT_ \rangle$ ‘drink’, its TAM is imperfective, its polarity is negative, its voice and stem-internal aspect are simplex, its subject is third person plural (‘they’), and its object is third person singular masculine (‘him/it’).

The cascade of FSTs that are composed to form the Amharic lexical noun FST is similar, except that it includes a separate lexicon of simple noun stems such as ጢት (in the file `Am/n_stem.lex`) alongside the embedded cascade that is responsible for deverbal noun stems such as ጣድረግ *madreg*.

The cascades for the noun and verb guesser FSTs are similar, except that they lack the stem and root lexicons that are part of the full lexical FSTs. There are also some restrictions on the root categories that the guesser FSTs recognize; this prevents massive ambiguity that would otherwise occur. For Amharic they only accept verb roots in the canonical categories (CCC and CCCC, where none of the consonants are ' , w, or y). Thus the Amharic verb guesser analyzer will analyze the word ትዠርግላችሁ, based on the imaginary root <Zrm>, but not the word ትዠራላችሁ, based on the imaginary root <Zr'>.

The lexical analyzers can deal with irregular words as well as those that obey the rules. One sort of irregularity is in the formation of verb stems. For example, the Amharic verb root <drg> has the alternate transitive imperfective stem *arg* (as in ያርጋል), alongside the regular stem *aderg* (as in ያደርጋል), and the Tigrinya verb root <whb> 'give' loses the *w* in most of its forms (ሃበ, ይህበ, ይሃበ, ሂበ). Irregular stems are stored in separate files, *Am/irr_stem.lex* and *Ti/irr_stem.lex*, and the analyzer tries these stems along a separate path in the FST in parallel with the path implementing the regular stems.

For Amharic, constraints on the voice that occurs with particular roots are included within the root file, *Am/vb_root.lex*, in the form of FS sets. For example, along with the root <qm_T> is the feature set {[vc=ps]; [vc=cs]; [vc=tr, as=it]}, representing the constraint that this root can only occur in the passive and causative voice and the iterative transitive voice.

b. DATA FILES

Data used by HORN MORPHO appear in three file types, distinguished by their extensions: *.fst*, files describing FSTs explicitly in terms of states and transitions; *.cas*, files specifying cascades of FSTs to be composed; *.lex*, files containing lists of roots, stems, or complete words, to be converted to FSTs. Any of these files may contain lines with comments; these begin with "#".

The only files needed to run the program are the FST files containing the fully composed analyzers and generators for verbs, nouns, and copulas. The other files are included in the distribution for those who would like to use the grammatical and lexical information that is contained in them for other purposes.

i. FST files

An FST (*.fst*) file implements one or more alternation rules or the morphotactics for a word class in the form of an FST. Each uncommented line in an FST line takes one of the following forms:

- -> STATE
Makes STATE the initial state of the FST.
- STATE ->
Makes STATE a final state of the FST.
- STATE1 -> STATE2 CONDITIONS FSset
Creates one or more transitions from STATE1 to STATE2. CONDITIONS has one of the following forms.

- [IO1; IO2; ...]

Each of IO* represents the condition on a transition from STATE1 to STATE2. It has one of the following forms:

- CHAR1:CHAR2

CHAR1 and CHAR2 are either characters from the alphabet of romanized Amharic or Tigrinya characters, a character set abbreviations (defined in the cascade file for this FST file), or nothing (the empty character).

Examples:

[i : y] input character *i* corresponds to output character *y*
 [X :] any input consonant (represented by the character set abbreviation “X”) corresponds to nothing in the output
 [:] no input character is consumed and no output generated on this transition

- CHAR

CHAR is either a character from the alphabet or a character set abbreviation. This represents identical input and output characters.

Examples:

[i] input and output characters are both *i*
 [X] input and output characters are the same consonant

- <CHARS:>

CHARS is a sequence of characters from the alphabet. This represents a sequence of states, each with a transition into it with one of the characters as the input character, no character as output character.

Example:

STATE1 -> STATE2 <et:>
 equivalent to
 STATE1 -> STATE1.2 [e:]
 STATE1.2 -> STATE2 [t:]

FSset, which is optional, represents any FS constraints which apply to the transition. Each FS has the form [FV1, FV2, ...]. Each FV* represents a single feature-value pair. It has one of these forms

- +FEATURE, -FEATURE

These are equivalent to FEATURE=True and FEATURE=False respectively.

- FEATURE=VALUE

Values is either a string or a full FS.

If there are multiple FSs in the set, they are separated by semicolons. The interpretation of multiple FSs is that at least one of the FSs must unify with the current accumulated FS at the point an attempt is made to traverse the transition.

- STATE1 -> STATE2 >>FST<<

FST is the name of an FST file (without the .fst extension). The FST described in this file is to be inserted between STATE1 and STATE2.

- STATE1 -> STATE2 +LEX+

LEX is the name of a lexical file (without the `.lex` extension). The FST represented by the list of roots, stems, or words in the file is inserted between STATE1 and STATE2.

HORN MORPHO currently has no facility for compiling FSTs from abstract descriptions of linguistic rules or from regular relations, as is possible in most other FST packages, such as OpenFst (<http://www.openfst.org/>) and the Xerox finite state tools (<http://www.stanford.edu/~laurik/fsmbook/home.html>). In later versions of the program, this capability may be added.

ii. Cascade files

A cascade file (`.cas`) gives a list of FST to be composed to create a single FST. Each of the FSTs is represented by the name of an FST or lexical file. Each of these appears on a separate line without its extension and surrounded by “>...<” for FST and “+...+” for lexical files.

Before the list of files, a cascade file may also contain a list of character set abbreviations. Each abbreviation is a character or sequence of characters (for example, `x`, `v`, `!x`, `JJ`). The specification for each abbreviation appears on a separate line. It begins with the abbreviation, is followed by “=” and the list of characters in the set, separated by commas, and surrounded by “{ }”, for example,

(13) `JJ = {d, l, n, s, t, T, z}`

iii. Lexical files

A lexical file (`.lex`) contains a list of lexical forms (roots, stems, or complete words) which are to be converted to an FST that accepts the forms as sequences of input characters. Each uncommented line has one of the following forms.

- FORM

FORM is a string representing a root, stem, or word. It is incorporated into the FST as a sequence of states joined by transitions with identical input and output characters.

- FORM ROOT FSset

FORM is a string representing a root, stem, or word. ROOT is the root that is to be associated with FORM. In this case, the FORM is incorporated into the FST as a sequence of states joined by transition with different input and output characters; that is, FORM as input results in ROOT as output. FSset specifies a set of FS constraints that are associated with FORM. FSset is optional. If FSset appears, and no separate root is to be specified, the second position is filled by the empty string “. An example, from the file `Am/irr_stem.lex`:

(14) `tegenaNt gN* [tm=ger,vc=ps,as=it]`

This indicates that the form *tegenaNt* is the passive, iterative, gerundive stem of the Amharic verb with the root `<gN*>` አገኘ. (This is irregular because we would normally expect *tegeNaNt*.)

Roots which are regular but which fail to occur with certain voice features are also notated with FS constraints in the lexicon file where they appear. An example, from the file containing regular Amharic verb roots, `Am/vb_root.lex`:

(15) `ds_t ' ' [vc=ps];[vc=cs];[v=man,pos=n]`

This indicates that the Amharic root <ds_t> ተጸሰተ occurs only in the passive and causative voices or the manner form for nouns.

8. Use

a. STARTING THE PROGRAM

To use HORN MORPHO, you must be running the Python interpreter. There are several ways to do this; you can run Python as a self-standing program or from within an integrated development environment, such as IDLE, the one that comes with the Python distribution. If you want to use the capability of the program to process Ge'ez characters within the Python interpreter (only possible with Linux and MacOS), you should not use IDLE because it currently fails to handle Unicode characters properly. Before you go any further, be sure your computer has version 2.5 or 2.6 of Python. If not, you can download version 2.6 from the Python site: <http://www.python.org/download/>. (The program will not run under Python 3.)

To start up Python on computers running Unix (including Macintosh computers) or Linux, you type `python` in a shell. On computers running Windows, you start the interpreter either by selecting Run in the Start menu and then typing `python`, by double-clicking on the Python application icon, or by typing `python` in a command prompt window. (If the third option fails, you will need to change your `path` variable so that it knows where to find Python. How to do this is beyond the scope of this document.)

If you (or a system administrator) have installed HORN MORPHO on your computer, you can run Python from anyplace in the file system. If not, you need to run it in the `HornMorpho-1.0` directory. You can either change to this directory in the shell before you start up Python, or you can change directories within Python after you start it up. To change directories within Python, first import the `os` module, then use the `os.getcwd()` command to see where you are, and the `os.chdir()` command to move around. For example,

```
>>> import os
>>> os.chdir('../..../Projects/HornMorpho1.0')
>>> os.getcwd()
'/Users/gasser/Projects/HornMorpho1.0'
```

On a Windows machine, you would use “\\” instead of “/” for the directory separator.

If HORN MORPHO is installed on your computer or you are in the `HornMorpho-1.0` directory, you should be able to import all the files you need.

If the `import` statement fails, you will probably see this message: `ImportError: No module named 13`. This happens because the path that the interpreter searches doesn't include the current directory. To fix this, do the following:

```
>>> import sys; sys.path.append('')
```

Now the `import` statement should work.

HORN MORPHO has four functions. Of these the most useful for most users will be `anal_file`, which analyzes the Amharic or Tigrinya words contained in one file, writing the analysis to another file. If you only want to use this function, you can skip to section d. The other functions analyze or

generate individual words; by default these expect words to be written in Ge'ez characters. That is, the Python interpreter must be able to handle Unicode, both as input and output. As far as I can tell, this is not yet possible under Windows, but it is straightforward under Linux and MacOS. The next section explains what you need to know to make the Python interpreter deal with Ge'ez under these operating systems.

b. GE'EZ CHARACTERS IN THE PYTHON INTERPRETER

When you start Python in a shell, the characters that Python can recognize and print out depend on the character encoding of the terminal program you are running. You need to make sure that the encoding is set to UTF-8 in order for Ge'ez characters to show up properly in the Python interpreter.

In Linux, UTF-8 is usually the default, in which case you won't have to set anything. If you do have to change the setting, what you do will depend on the terminal program you're running. In RedHat Linux, for example, you would be running a program called Terminal, which has a menu called "Terminal" with a menu item called "Set character encoding".

To set the character encoding in MacOS, start up the Terminal program; then go to the Preferences menu item (in the Terminal menu). You'll see different configuration of Terminal windows on the left; select the configuration that you'd like to set the character encoding for. Click on the Advanced tab. At the bottom of the window, under "International", you'll see a list titled "Character encoding". Select "Unicode (UTF-8)". Then close the Preferences window. Now when you open a new Terminal window (shell) with the configuration that you changed, it should be able to display Ge'ez characters.

You can test your shell without running Python by typing something like the following ("%" is the prompt).

```
% echo 'ፊተኛ'
ፊተኛ
```

Assuming you have a Unicode Ge'ez font on your computer, you should see what appears above.

c. ANALYZING A WORD

To analyze a single word, enter the following at the Python interpreter prompt:

```
l3.anal_word(language, word)
```

where *language* is a Python string representing the language you want to analyze, either 'am' or 'ti', and *word* is a word in Ge'ez characters and enclosed in quotation marks (either single or double). If you are using Windows or running Python in a shell that is not set up to handle Unicode characters, you can type the word in romanized form instead, setting the *non_roman* option to `False`.

```
l3.anal_word(language, word, non_roman=False)
```

In the romanization you should not include gemination or the vowel *I*.

If you have not called `anal_word` or one of the other functions to be described below for *language*, the program will first have to load the data it needs to perform the analysis. Because the data files are very large, it could take perhaps 30 or 40 seconds for this to complete.

After the data are loaded, the program checks to see whether the word is in its lexicon of unanalyzed words; in these cases, it just prints out the word.

```
(16) >>> l3.anal_word('ti', 'ፍብ')
      Word: ፍብ
```

The romanized alternative would be as follows.

```
(17) >>> l3.anal_word('ti', 'nab', non_roman=False)
      Word: nab
```

Some of these “unanalyzed” words could be analyzed into constituent morphemes, for example, the Amharic adverb **ይልቁገዎ**, but because they function as idiomatic semantic wholes, such an analysis would not be particularly useful.

If the word is not in the unanalyzed lexicon, the program checks its lexicon of pre-analyzed words, words which either have irregular morphology or are very common. If it is not there, the program attempts to analyze it using one of its FSTs, first with lexical FSTs for verbs and nouns that contain lists of noun stems and verb roots, then with “guesser” FSTs that attempt to guess an unknown stem or root based on the prefixes and suffixes and the shape of the possible stem.

If the program succeeds in analyzing a word, it will print out the root or stem, the citation form, and the grammatical analysis. Here are examples of the analysis of relatively complex verbs in both languages.

```
(18) >>> l3.anal_word('am', 'የማያስፈልጋትስ')
      Word: የማያስፈልጋትስ
      POS: verb, root: <fl_g>, citation: አስፈለገ
      subject: 3, sing, masc
      object: 3, sing, fem
      grammar: imperfective, causative, relative, negative
      conjunctive suffix: s
      >>> anal_word('ti', 'ብዙጋጥመና')
      Word: ብዙጋጥመና
      POS: verb, root: <gTm>, citation: አጋጠመ
      subject: 3, sing, masc
      object: 1, plur
      grammar: imperfective, reciprocal, transitive, relative
      preposition: bI
```

The first line in the analysis just repeats the word. The second line gives the part of speech, the root of the word, and the citation form. For verb root categories and their representation in HORN MORPHO, see Section 5.a.ii above. The citation form used here incorporates the voice and stem-internal aspect of the input word (see Section 5.a.i for a discussion of these grammatical features). Thus what appears here is **አስፈለገ** for the Amharic example and **አጋጠመ** for the Tigrinya example rather than the more basic forms **ፈለገ** and **ገጠመ**, which have the value `simplex` for both voice and stem-internal aspect. On the following lines is the grammatical analysis of the word. On the first two lines of the examples are the features of the verb’s subject and object. The next line lists miscellaneous grammatical features: imperfective tense/aspect/modality, reciprocal stem-internal aspect for the Tigrinya example, causative voice for the Amharic example, transitive voice for the Tigrinya, and `True` values for the `relative` feature for both languages and the `negative` feature for Am-

haric (see Section 5.a.ii and 5.a.iii for more on what these mean). The last line lists any prefixed or suffixed prepositions, conjunctions, or adverbs (see Section 5.a.iii for more on these).

Here is an example of the analysis of an Amharic noun that is not derived from a verb (more precisely, a noun that the program does not *know* is derived from a verb).

```
(19) >>> l3.anal_word('am', 'ከየኃላፊዎቻቸው')
Word: ከየኃላፊዎቻቸው
POS: noun, stem: ኃላፊ
    possessor: 3, plur, masc
    grammar: plural, definite, distrib(Iyye-)
    preposition: ke
```

Since there is no known verb root for the word, only the stem is given in the second line: ኃላፊ. The third line gives features of the possessor; it is third person plural. The fourth line gives miscellaneous grammatical properties—the noun is plural, definite and distributive (that is, it has the prefix እየ 'Iyye-)—and the fifth line gives any prepositional, conjunctive, or adverbial prefixes or suffixes, in this case the preposition ከ *ke*.

Here is an example of the analysis of an Amharic noun that is derived from a verb.

```
(20) >>> l3.anal_word('am', 'ባለጣጠናቀቃችን')
Word: ባለጣጠናቀቃችን
POS: infinitive, root: <Tnqq>, citation: አጠናቀቀ
    possessor: 1, plur
    grammar: reciprocal, transitive, negative
    preposition: be
```

The part of speech that is given in the second line of the analysis of a deverbal noun is either infinitive, agentive noun, instrumental noun, or manner noun; the root given is the verb root, and the citation form is that of the verb, incorporating the voice and stem-internal aspect of the input word (in the example, transitive voice and reciprocal aspect).

If `anal_word` relies on one of its guesser analyzers, the second line of the analysis is preceded by a question mark. In the following Tigrinya example, the verb guesser analyzer is called for because the verb root `<mnTl>` is not in the lexical analyzer's lexicon.

```
(21) >>> l3.anal_word('ti', 'እናመንጠለት')
Word: እናመንጠለት
?POS: verb, root: <mnTl>, citation: መንጠለ
    subject: 3, sing, fem
    grammar: perfective
    conjunctive prefix: Inna
```

If `anal_word` fails to analyze the word, then a question mark precedes the single line that is printed out.

```
(22) >>> l3.anal_word('ti', 'ፔፕሲ')
?Word: ፔፕሲ
```

If multiple analyses are possible, they are all printed out by `anal_word`.

```
(23) >>> l3.anal_word('am', 'ነገራችሁ')
Word: ነገራችሁ
POS: noun, stem: ነገር
    possessor: 2, plur
POS: verb, root: <ngr>, citation: ነገረ
    subject: 3, sing, masc
    object: 2, plur
    grammar: perfective
POS: verb, root: <ngr>, citation: ነገረ
    subject: 2, plur
    grammar: perfective
```

Amharic and Tigrinya have two verbs which behave like no others, the copula (ነው; እዩ, etc.) and the verb of existence (አለ, የለም; አሎ, የለን, etc.). HORN MORPHO outputs <ne> for Amharic and <y> for Tigrinya as the root of the copula and <al_e> as the root of the verb of existence. Here are two examples.

```
(24) >>> l3.anal_word('am', 'አይደለችም')
Word: አይደለችም
POS: copula, root: <ne>
    subj: 3, sing, fem
    negative
>>> l3.anal_word('am', 'የሌለባችሁ')
Word: የሌለባችሁ
POS: verb, root: <al_e>, citation: አለ
    subject: 3, sing, masc
    object: 2, plur, prep:-b-
    grammar: present, relative, definite, negative
>>> l3.anal_word('ti', 'ድዮም')
Word: ድዮም
POS: copula, root: <y>
    subject: 3, plur, masc
    grammar: yes/no
>>> l3.anal_word('ti', 'ዘየብለይ')
Word: ዘየብለይ
POS: verb, root: <al_e>, citation: አሎ
    subject: 3, sing, masc
    object: 1, sing
    grammar: present, relative, negative
```

If you would like to suppress the root/stem, citation form, and/or grammatical analysis that are printed out by `anal_word`, you can do this by specifying `False` for the options `root`, `citation`, and/or `gram`.

```
(25) >>> l3.anal_word('am', 'ቢያስጨንቁክኛው', root=False, gram=False)
Word: ቢያስጨንቁክኛው
POS: verb, citation: አስጨነቀ
>>> l3.anal_word('ti', 'ዝፈልሰኛ', gram=False)
Word: ዝፈልሰኛ
?POS: verb, root: <flsf>, citation: ፈልሰፈ
```

d. ANALYZING A FILE

To analyze all the words in a file and write the analyses to another file, enter the following at the Python interpreter prompt.

```
l3.anal_file(language, input_file, output_file)
```

where *language* is again either 'am' or 'ti', *input_file* is a file name or a path to a file containing Amharic or Tigrinya in Unicode Ge'ez, and *output_file* is a file name or a path to an output file. All must be surrounded by quotation marks. The file at *input_file* must exist; otherwise the program reports an error. The file at *output_file* need not exist, but if *output_file* is a path to a file in a directory, the directory must exist, and the user must have permission to write to a file there.

The function *anal_file* segments the input file into words and attempts to analyze each of these, using *anal_word*. It writes the result of the analyses to the output file. Punctuation marks, other than '-', which can be word-internal in Amharic and Tigrinya, are separated and treated as words. No attempt is made to analyze these or any words that contain non-Ge'ez characters or Ge'ez numerals.

Assume that following sentence (the first sentence from the new preface to Abe Gubegna's novel **አልወለድም**) is contained in the file *am.txt*, which is located in a directory called *Data* in the *l3* directory under the directory where Python is running (this file is included with the distribution in that location).

ይህ መጽሐፍ የዛሬ 01 ዓመት ገደማ በደንቡ ምርመራ አልፎ ታትሞ በወጣ ጊዜ ታላቅ ችግር ፈጥሮብኝ ነበር ።

Then the following will analyze this sentence, writing the analysis to a file in the same directory called *alweledm_out.txt*. (You would use '\\' instead of '/' in the paths in Windows.)

```
(26a) >>> l3.anal_file('am', 'am.txt', 'l3/Data/alweledm_out.txt')
Analyzing words in l3/Data/am.txt
Writing analysis to l3/Data/alweledm_out.txt
```

The contents of *alweledm_out.txt* are then as follows.

```
(26b) Word: ይህ
      POS: noun, stem: ይህ
      Word: መጽሐፍ
      POS: noun, stem: መጽሐፍ
      Word: የዛሬ
      POS: noun, stem: ዛሬ
      grammar: genitive
      POS: noun, stem: ዛሬ
      possessor: 1, sing
      grammar: genitive
      Word: 01
      Word: ዓመት
      POS: noun, stem: ዓመት
      Word: ገደማ
      POS: noun, stem: ገደማ
      Word: በደንቡ
```

POS: noun, stem: **ደንብ**
 possessor: 3, sing, masc
 preposition: be
 POS: noun, stem: **ደንብ**
 grammar: definite
 preposition: be
 Word: **ምርመራ**
 POS: noun, stem: **ምርመራ**
 Word: **አለፈ**
 POS: verb, root: <'lf>, citation: **አለፈ**
 subject: 3, sing, masc
 grammar: gerundive
 POS: verb, root: <'lf>, citation: **አለፈ**
 subject: 3, sing, masc
 grammar: gerundive, transitive
 Word: **ታተመ**
 POS: verb, root: <'t_m>, citation: **ታተመ**
 subject: 3, sing, masc
 grammar: gerundive, passive
 Word: **በወጣ**
 POS: verb, root: <wT'>, citation: **ወጣ**
 subject: 3, sing, masc
 grammar: perfective, relative
 preposition: be
 Word: **ጊዜ**
 POS: noun, stem: **ጊዜ**
 Word: **ታላቅ**
 POS: noun, stem: **ታላቅ**
 POS: verb, root: <lqq>, citation: **አላቀቀ**
 subject: 3, sing, fem
 grammar: jussive/imperative, reciprocal, transitive
 Word: **ችግር**
 POS: noun, stem: **ችግር**
 Word: **ፈጥሮብኝ**
 POS: verb, root: <fTr>, citation: **ፈጠረ**
 subject: 3, sing, masc
 object: 1, sing, prep:-b-
 grammar: gerundive
 Word: **ነበር**
 Word: **#**

e. GENERATING A WORD

You can also use HORN MORPHO to generate words, given their root or stem and grammatical features. In order to do this, you will need to be familiar with the way verb roots and grammatical features are represented in the program.

In its simplest form, the generation function looks like this:

```
13.gen(language, root/stem)
```

where *language* is again either 'am' or 'ti' and *root/stem* is a romanized noun stem or verb. In this form, the function generates the wordform that conforms to a default set of features associated with the part-of-speech for the word. For verbs, the default grammatical structure is as follows.

- subject: third person masculine singular; no object
- tense/aspect/mood: perfective
- voice: simplex; stem-internal aspect: simplex
- affirmative, non-relative
- no prepositional, conjunctive, or adverbial affixes

Here is two examples of verbs with the default structure.

```
(27) >>> l3.gen('am', "mWl' ")
      ጠላ
      >>> l3.gen('ti', "gWyy")
      ኀየየ
```

That is, ጠላ *mol_a* is the third person masculine singular perfective form of the verb with the root <*mWl'*>. Note that the root is surrounded by quotation marks, which in the first case must be double because the root contains the single quotation character. For more on the representation of verb roots in HORN MORPHO, see Section 5.a.ii.

If you are using Windows or running Python in a shell that is not set up for Unicode, you can have these functions produce romanized output by setting the `non_roman` option to `False`.

```
(28) >>> l3.gen('am', "mWl' ", non_roman=False)
      mola
```

For nouns, the default grammatical structure is as follows.

- no possessor
- singular number, indefinite
- no prepositional prefix, genitive prefix, or conjunctive/adverbial suffix
- not derived from a verb

Thus if you give the generation function a noun stem and no grammatical specification, the function just returns the stem unchanged.

```
(29) >>> l3.gen('am', "meng^st")
      መንግሥት
```

To specify grammatical structure other than the default, you put this after the root or stem (with a comma in between), using abbreviated versions of the feature structure representation described in Sections 4.b. and 7.b.i. above. Each feature-value pair that you include represents a change in the default set of features. Feature structures are enclosed in brackets and surrounded by quotation marks. A simple feature-value pair is represented by the feature and the value separated by an equals sign. For example, if you want the passive form of a verb, with no other features in the default changed, you would set the `voice` feature (`vc`) to `passive` (`ps`).

```
(30) >>> l3.gen('am', "mWl' ", '[vc=ps]')
      ተጠላ
      >>> l3.gen('ti', 'HSb', '[vc=ps]')
      ተሐጽበ
```

Values of `True` and `False` can be represented using `+` or `-` signs before the feature name. For example, to make a noun definite (`def`) and plural (`plr`), you would do the following.

```
(31) >>> l3.gen('am', "meng^st", '[+plr,+def]')
መንግሥታቱ
```

To specify the subject or object of a verb, you will need a nested feature structure because the value of the subject and object features is itself a feature structure. For example, to make the subject (`sb`) of a verb second person (`p2`) singular feminine (`fem`) and the object (`ob`) third person plural (`plr`), you would do the following.

```
(32) >>> l3.gen('am', "mWl", '[sb=[+p2,+fem],ob=[+plr]]')
ሞላሻሻው
>>> l3.gen('ti', 'HSb', '[sb=[+p2,+fem],ob=[+plr]]')
ሐጸብክሎም
```

Object suffixes may also be indirect. To generate an indirect object in Tigrinya, you use the object feature `+prp`. For example, you have to specify one of the features `+b` or `+l`, depending on the indirect object prefix.

```
(33) >>> l3.gen("mWl", '[sb=[+p2,+fem],ob=[+plr,+l]]')
ሞላሻሻው
>>> l3.gen('ti', 'HSb', '[sb=[+p2,+fem],ob=[+plr,+prp]]')
ሐጸብክሎም
```

To generate an Amharic noun that is derived from a verb, use `gen` with a verb root, and, in addition to the part-of-speech (`pos=n`), specify which category of deverbal noun you want as the value of the `v` feature: infinitive (`inf`), agentive noun (`agt`), instrumental noun (`ins`), or manner noun (`man`). For all but the manner noun, you may also specify values of the voice (`vc`) and stem-internal aspect (`as`) features.

```
(34) >>> l3.gen('am', "sdb", '[pos=n,v=agt,vc=cs,as=rc]')
አሳዳቢ
```

The generation function does not normally use the guesser FSTs; if you'd like to force the use of the guesser FSTs, use the option `guess=True`.

```
(35) >>> l3.gen('am', 'kongo', '[pp=be]')
This word can't be generated!
>>> l3.gen('am', 'kongo', '[pp=be]', guess=True)
በኮንጎ
```

Amharic noun stems must be romanized and must include gemination, even though this is not indicated in the standard orthography. The vowel *I* should not be included.

```
(36) >>> l3.gen('am', 'wddr', '[+gen, poss=[+p1,+plr]]')
This word can't be generated!
>>> l3.gen('am', 'wdd_r', '[+gen, poss=[+p1,+plr]]')
የውድድራችን
```

To see the full list of features and possible values for a language and part-of-speech, use the function `get_features`, which returns a Python dictionary.

```
(37) 13.get_features('ti', 'v')
      {u'vc': [u'ps', u'smp', u'tr'], u'yn': [True, False], 'pos':
      ['v'], u'as': [u'smp', u'rc', u'it'], u'sub': [True, False],
      u'pp': [u'sIle', u'Inte', u'nab', u'nI', u'kab', u'bI', u'kem',
      u'ab'], u'd': [True, False], u'neg': [True, False], u'ob':
      {u'p2': [True, False], u'p1': [True, False], u'plr': [True,
      False], u'xpl': [True, False], u'fem': [True, False], u'prp':
      [True, False]}, u'tm': [u'imf', u'j_i', u'ger', u'prf', u'prs'],
      u'rel': [True, False], u'cj2': [u'n', u's', u'ke', u'do',
      u'Immo'], u'sb': {u'p2': [True, False], u'p3': [True, False],
      u'fem': [True, False], u'p1': [True, False], u'plr': [True,
      False]}, u'cj1': [u'mIs', u'InkI', u'kI', u'Inte', u'mI',
      u'Inna', u'nIKI', u'nI']}
```

For reference, here is the full list of abbreviations for features and values that you may need is as follows:

- subject: sb; object: ob; possessor (nouns): poss
- 1st person: p1; 2nd person: p2; feminine: fem; plural: plr; formal: frm; prepositional object: l, b (Amharic), prp (Tigrinya)
- tense/aspect/mood: tm; perfective: prf, imperfective: imf, jussive/imperative: j_i, gerundive: ger
- voice: vc; simplex: smp, passive-reflexive: ps, transitive: tr, causative (Amharic): cs
- stem-internal aspect: as; simplex: smp, reciprocal: rc, iterative: it
- genitive (nouns): gen; definite (Amharic): def; relative: rel; negative: neg; accusative (Amharic): acc; distributive (Amharic, *Iyye*): dis
- auxiliary (Amharic): ax; alle: al
- prefix conjunction: cj1; suffix conjunction (verbs): cj2; suffix conjunction (nouns): cnj
- preposition: pp

More examples:

```
(38) >>> 13.gen('am', "ngr", '[pp=ke,tm=imf,vc=ps,as=it,cj2=m]')
      ከሚከገርም
      >>> gen('am', 'ne', '[+neg, sb=[+p1,+plr]]')
      አይደለም
      >>> 13.gen('am', 'brkt', '[pos=n,v=ins, pp=ke, cnj=m, +def]')
      ከመበርከቻውም
      >>> 13.gen('ti', 'n|qTqT', '[vc=ps, tm=imf, sb=[+p1,+plr]]')
      ንንቅጥቅጥ
      >>> 13.gen('ti', 'gdf', '[tm=j_i,+neg,sb=[+p2],ob=[+plr],vc=ps,as=rc]')
      አይትጋደፎም
      >>> 13.gen('ti', 'qS_1', '[sb=[+p2,+plr], ob=[-plr], cj1=kI, tm=imf]')
      ከትቅጽሉዎ
```

9. Limitations

a. GENERAL USABILITY

In HORN MORPHO speed is sacrificed in favor of coverage. Because all possible analyses are returned, the search is non-deterministic; after an analysis has been found, the program backtracks and looks for another, until there are no more options. This can take considerable time. Generation is even slower, especially for relatively long verbs; this is due to the placement of feature structure constraints at the end of roots, a problem that Amtrup (2003) discusses.

A related problem that is less easy to fix concerns the program's inability to rank different analyses by their likelihood. Consider the Amharic word ታላቅ. The usual interpretation of this word is as an adjective (called "noun" in HORN MORPHO). But there is a much less likely possibility, that the word represents the third person singular feminine jussive form of the verb <ḥqq> in transitive voice and reciprocal aspect, an interpretation that might be translated 'let her make someone let go of something'. HORN MORPHO returns both analyses, with no indication of preference. The problem is worse with the Amharic noun guesser analyzer because of the different ways in which noun suffixes can be segmented when the stem is unknown. The guesser analyzer is called in the following Amharic example because ዲፕሎማት is not in the program's lexicon of noun stems.

```
(38) >>> l3.anal_word('am', 'ዲፕሎማቶችን')
Word: ዲፕሎማቶች
?POS: noun, stem: ዲፕሎማቶች
  possessor: 1, plur
?POS: noun, stem: ዲፕሎማት
  possessor: 1, plur
  grammar: plural
?POS: noun, stem: ዲፕሎማቶች
  grammar: accusative
?POS: noun, stem: ዲፕሎማታ
  possessor: 1, plur
  grammar: plural
```

The program would perhaps be usable to a wider audience if more of the interface, including the grammatical terminology, were in Amharic or Tigrinya rather than English. The author will need the help of native speakers to make this improvement in future versions of HORN MORPHO.

b. MORPHOLOGICAL ISSUES

A significant weakness of the Tigrinya component of HORN MORPHO is that it only handles verbs. Tigrinya nouns are more complex than Amharic nouns because of their plural formation, and there is apparently no digitized dictionary that lists plural forms along with the corresponding singular forms.

For both languages common words are stored in a dictionary of forms that the program returns un-analyzed. In the current version, no part-of-speech or other information is provided for these words.

An important category of Amharic and Tigrinya words (in fact one of the important defining features of the Ethio-Semitic language family) that HORN MORPHO does not currently analyze are

those that Baye (2000 E.C.) calls የአደራረግ አዕማድ in Amharic. These are derived from verbs and behave semantically like verbs (or adverbs) but are not conjugated themselves; instead they combine with the verbs አለ or አደረገ in Amharic, በለ or አበለ in Tigrinya. For example, ስብር አለ, እቱ በለ.

In general, HORN MORPHO handles Amharic nouns and adjectives less completely than verbs. Specifically, the program does not yet know about the following.

- a. Noun plurals formed with the prefix *In_e*, for example, እነከበደ.
- b. Adjective plurals formed through reduplication, for example, ትንንሽ.
- c. Noun and adjective derivation from other other nouns or adjectives through the suffixes *-n_et*, *-m_a*, *-N_a*, for example, ነጻነት, ውኃማ, ደንበኛ.
- d. Noun and adjective derivation from verbs beyond the four categories discussed above, for example, ክብደት (from <*kbd*>), ሞጥሟጣ (from <*mWTmWT*>), ፍጹም (from <*fS_m*>).

c. LEXICAL ISSUES

The Amharic root and stem lexicons in the program were derived mainly from the Amharic-English dictionary of Amsalu (1979 E.C.), which is available under the Creative Commons Attribution-Noncommercial 3.0 United States License at <http://nlp.amharic.org/resources/lexical/word-lists/dictionaries/amsalu-aklilu-amharic-english-dictionary/>. The noun stem lexicon is far from complete; in particular it is missing all personal names and almost all place names.

The Tigrinya verb root lexicon in the program is limited to verb roots that could be extracted from Efrems's (2009) online Tigrinya-English dictionary (in the version as of February 2008) and a few others that have been added by hand; there are currently about 600 entries. Thus the guesser analyzer must be relied on much of the time.

Intransitive and transitive verb roots are not distinguished in the lexicon, so the program analyzes a number of words that are not actually possible, such as Amharic *ሞተኝ or Tigrinya *ሞይቱኒ. There is also nothing that prevents verbs in passive/reflexive voice from taking direct objects, so an impossible word like Amharic *ተመታኋት or Tigrinya *ተሃረመዖ is also analyzed as though it were grammatical. (Note that some formally passive Amharic and Tigrinya verbs *can* take object suffixes: ተሰማኝ, ተቀበልኩት; ተሰሚዑኒ, ተቐብለዮ; etc.) Within the framework used to implement HORN MORPHO, it is straightforward to include constraints like these in the lexicon; the root is simply annotated with a feature structure constraint that limits the possible values for the *obj* feature. A goal in future work is to have this happen automatically in response to what possibilities occur in Amharic and Tigrinya corpora.

d. ORTHOGRAPHIC AND PHONOLOGICAL ISSUES

There is some variation in the spelling of Amharic words, and the program handles some of the options, for example, ሰጧኞሁ ~ ሰጡአኞሁ. Other variation is not covered. For example, only the first of these alternative spellings is analyzed by the program: ሰረቅሽው ~ ሰረቅሺው.

Amharic orthography includes two ways to represent the phoneme *s*, two ways to represent the phoneme *S*, three ways to represent the phoneme *h*, and two ways to represent vowels with no pre-

ceding consonant or a glottal stop. Although lexicographers have attempted to standardize the spellings of words with these sounds, and there is some agreement for common words, there is still considerable variation in the selections that writers make to spell these sounds. Ideally a morphological analyzer would handle all of this variation, but HORN MORPHO currently does not. Although it does include alternate spellings for a few roots and stems, for the most part, a word must conform to the single spelling the program is familiar with in order to be analyzed.

```
(39) >>> l3.anal_word('am', 'ዑሀ')
      Word: ዑሀ
      POS: noun, stem: ዑሀ
      >>> l3.anal_word('am', 'ዑኃ')
      ?Word: ዑኃ
```

Within Tigrinya writing, there appear to be two different competing tendencies regarding the use of the characters that have the same pronunciation (note that in Tigrinya the **አ** and **ዐ** series are *not* pronounced the same). One is to attempt to standardize the spellings of words with these sounds based on the way they, or related words, were written in Ge'ez itself. Another is to forego the use of the less common characters altogether, that is, to avoid the **ኅ**, **ሠ**, and **ፀ** sets. HORN MORPHO follows this second trend and thus fails to handle any words containing characters in these sets.

```
(40) >>> anal_word('ti', 'ተሰርሐ')
      Word: ተሰርሐ
      POS: verb, root: <srH>, citation: ተሰርሐ
      subject: 3, sing, masc
      grammar: perfective, passive
      >>> anal_word('ti', 'ተሠርሐ')
      ?Word: ተሠርሐ
```

Future versions of the program will have an option allowing users to choose to handle characters in the **ኅ**, **ሠ**, and **ፀ** sets for Tigrinya.

Of course HORN MORPHO also fails to handle genuine misspellings, including some that are quite common, such as the substitution of **ዉ** for **ዑ**.

References

- ባዩ ይማም። 2000 ዓ.ም.። የአማርኛ ሰዋሰው፣ የተሻሻለ ሁለተኛ እትም። አዲስ አበባ፣ እሴኒ ማተሚያ ቤት።
- አምሳሉ አክሊሉ። 1979 ዓ.ም.። አማርኛ - እንግሊዝኛ መዝገበ ቃላት Amharic-English Dictionary. አዲስ አበባ፣ ኩራዝ አሳታሚ ድርጅት።
- Amanuel Sahle (1998). ሰዋሰው ትግርኛ ብሰፊሑ. *A comprehensive Tigrinya grammar*. Lawrenceville, New Jersey: Red Sea Press.
- Amtrup, J. (2003). Morphology in machine translation systems: efficient integration of finite state transducers and feature structure descriptions. *Machine Translation*, 18, 213-235.
- Beesley, K.R., and Karttunen, L. (2003). *Finite state morphology*. Stanford, California: CSLI Publications.
- Bender, M.L. and Hailu Fulass. (1978). *Amharic verb morphology*. East Lansing, Michigan: African Studies Center, Michigan State University.

- Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge: Cambridge University Press.
- Cohen-Sygal, Y., and Wintner, S. (2006). Finite-state registered automata for non-concatenative morphology. *Computational Linguistics*, 32, 49-82.
- Efrem Zacarias (2009). Memhir.org Dictionaries (English-Tigrinya, Hebrew-Tigrinya dictionaries). <<http://www.memhr.org/dic/>>
- Gasser, M. (2006). *How language works*. <<http://www.indiana.edu/~hlw/>>.
- Gasser, M. (2009). Semitic morphological analysis and generation using finite state transducers with feature structures. *Conference of the European Chapter of the Association for Computational Linguistics*, 12,
- Johnson, C.D. (1972). *Formal aspects of phonological description*. The Hague: Mouton.
- Kaplan, R.M., and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20, 331-378.
- Karttunen, L., Kaplan, R.M., and Zaenen, A. (1992). Two-level morphology with composition. *Proceedings of the International Conference on Computational Linguistics*, 14, 141-148.
- Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production*. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.
- Leslau, W. (1941). *Documents tigrigna: grammaire et textes*. Paris: Libraire C. Klincksieck.
- Leslau, W. (1995). *Reference grammar of Amharic*. Wiesbaden: Harrassowitz.
- Saba Amsalu and Girma A. Demeke. (2006). Non-concatenative finite-state morphotactics of Amharic simple verbs. *ELRC Working Papers*.
- Yitna Firdyiwek and Yaqob, D. (1997). The System for Ethiopic Representation in ASCII. <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.3191>>.