research group for human language technology and the democratization of information

ANTIMORFO 1.1 User's Guide

Michael Gasser

Indiana University, School of Informatics and Computing
gasser@cs.indiana.edu

28 September, 2010

1. Introduction

ANTIMORFO is a Python program that analyzes and generates morphologically complex words in two of the main languages of the Andes, (Cuzco) Quechua and Spanish (*Anti* is the Quechua word for Andes). ANTIMORFO uses finite state transducers augmented with grammatical constraints in the form of feature structure descriptions. For more on finite state morphology, see Beesley and Karttunen (2003); for a description of the specific approach used in ANTIMORFO, see Gasser (2009).

In the rest of this document, it is assumed that you have at least a basic knowledge of the languages covered and of basic grammatical terminology. It will also help if you know something about Python.

2. Installation

ANTIMORFO requires Python 3.0 or 3.1. It will not run under Python 2. If you haven't already done it, uncompress the file that you downloaded. This will yield a directory called `AntiMorfo-1.1`, which contains all of the files that you need to run ANTIMORFO.

You don't have to install the program to use it. In fact since the data for Quechua is likely to change considerably in the coming months, you may want to wait and just put the directory in a handy place and run the program from there.

To install ANTIMORFO, change to the `AntiMorfo-1.1` directory, and do the following.

(1) `python setup.py install`

making sure you're using Python 3.

3. Use

a. Starting the program

If you have installed ANTIMORFO, start the program by doing the following.

(2)
```
>>> import l3
```

If you have not installed it, change to the `AntiMorfo-1.1` directory and run the `import` statement above. If this fails, you probably don't have the current directory in your search path. To fix this, do the following.

(3)
```
>>> import sys; sys.path.append('')
```

Now the `import` statement should work.

ANTIMORFO has three kinds of functions, for analyzing individual words, for analyzing all of the words in a file, and for generating words.

b. Analyzing a word

To analyze a single word, enter the following.

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

where *`language`* is an identifier for one of the supported languages, `'Español'`, or `'es'` for Spanish and `'Quechua'` or `'qu'` for Quechua, and *`word`* is a word in the form of a Python string.

The program first loads the morphological data for the language if this has not already happened. It then attempts to analyze the word using the built-in finite state transducers that encode lexical and grammatical information about words. Note that the program only has knowledge about verbs for Spanish; it knows about all categories of words in Quechua. If the program can analyze the word, it prints out the root or stem of the word and a description of the word's grammatical features. Here are examples for both of the supported languages. Note that in the case of ambiguity, all analyses are printed out.

(5)
```
>>> l3.anal_word('es', 'volvieron')
Cargando datos morfológicos para español ...
Palabra: volvieron
CG: verbo, raíz: <volver>
 pretérito; subjeto: 3, plur

>>> l3.>>> anal_word('qu', 'piskawankiku')
Palabra: piskawankiku
CG: verbo, raíz: <piska>
 español: tomar, coger, prender, asir, agarrar
 subjeto: 2 sing
 complemento directo: 1 plur excl
 TAM: presente/futuro
CG: verbo, raíz: <piska>
 español: tomar, coger, prender, asir, agarrar
 subjeto: 2 plur
 complemento directo: 1 plur excl
 TAM: presente/futuro
```

For both languages, the part-of-speech ("categoría gramatical") is printed out, followed by a representation of the word's root or stem ("raíz"). For Spanish, this is the infinitive of the input verb; for Quechua it is the root or stem of the noun, adjective, or verb (see Section 4.b below for details). For Quechua, this line is followed by a series of Spanish glosses for the root or stem of the Quechua word (not for the word itself). Next are one or more lines of grammatical features. For more on these features, see Section 4 below.

Instead of printing out a description of the grammatical properties of the word, you can have `anal_word` return the raw output of the transducers by setting the optional `raw` parameter to `True`. The output consists of a list of tuples, each consisting of a (string) root or stem and a feature structure description.

(6)
```
>>> l3.anal_word('qu', 'piskawan', raw=True)
[(u'piska', [asp=[contin=None, -prg], conj=[-add, -cop, -seq],
cs=None, der=[-fct, -inch, -lim, -rcp, -rfl], m=[AGR1=u'wa',
AGR2=u'n'], ob=[+p1, -p2, -pl, +xpl], pos=u'v', pos2=u'v',
prag=[-dbt, -emot, -emph, ev=None, -intj, -nonassr, -progn,
-resp, -top, -xplc], rpos=u'v', sb=[-p1, -p2, -pl],
t=[es='tomar, coger, prender, asir, agarrar'], tam=[-cd, -ft,
-pp, -ps], +tr])]
```

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

(7)
```
>>> l3.anal_word('qu', 'pepsi')
?Palabra: pepsi
```

c. Analyzing a file

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

(8)
```
l3.anal_file(language, input_file, output_file)
```

where *`language`* is an identifier for one of the supported languages, *`input_file`* is a file name or a path to a file, and *`output_file`* is a file name or a path to an output file, both in the form of strings.

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 are separated and treated as words.

d. Generating a word

To generate words using AntiMorfo, you will need to be familiar with the way roots and grammatical features are represented in the program. With no grammatical features provided, the grammatical structure of the word that is generated conforms to a default set of features for the language. For Spanish, this is the third person singular present of verbs (the only part-of-speech currently handled). For Quechua it is the bare form (singular, nominative) of nouns.

To generate a word in its default form, enter the following.

(9)
```
>>> l3.gen(language, root_stem)
```

where `language` is an identifier for one of the supported languages, `part_of_speech` and `root_stem` is a string representing the root or stem of a word. Here are examples from both languages.

(10)
```
>>> l3.gen('es', 'aprobar')
aprueba
>>> l3.gen('qu', 'wasi')
wasi
```

To specify grammatical structure other than the default, you put this after the root or stem. It takes the form of a feature structure, consisting of feature-value pairs. Each feature-value pair that you include represents a change in the default set of features. Feature structures are represented as strings of bracketed expressions. A simple feature-value pair is represented by the feature and the value separated by an equals sign. For example, if you want to generate a Quechua verb, you have to specify the part-of-speech, in order to override the default (noun).

(11)
```
>>> l3.gen('qu', 'piska', '[pos=v]')
piskan
```

This produces the third person singular form of the verb (with no direct object suffix), the default person and number for the subject.

The values `True` and `False` can be represented using `+` or `-` signs before the feature name. For example, to make the Quechua noun *wasi* plural (`pl`), you would do the following.

(12)
```
>>> l3.gen('qu', 'wasi', '[+pl]')
wasikuna
```

To specify the subject or object of a verb (or change the default values), 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 the Quechua verb *piskay* (root *piska*) first person (`p1`) singular and the object (`ob`) second person (`p2`) plural (`pl`), you would do the following.

(13)
```
>>> l3.gen('qu', 'piska', '[pos=v,sb=[+p1],ob=[+p2,+pl]]')
piskaykichis
```

The features and values that make up the description of the grammatical structure of a word are based on a particular theory of the morphology of the language in question; this is discussed informally in Section 4 below. It is also possible to get an idea of how the structure is represented by looking at the features and values that are selected from. To see the features and their possible values for a given language, do the following.

(14)
```
>>> l3.get_features(language)
```

The function `get_features` returns features and values as a Python `dict`. Here's what gets returned for Spanish.

(15)
```
>>> l3.get_features('es')
{u'as': {u'cnt': [True, False], u'prf': [True, False]}, u'refl':
[True, False], u'tam': {u'prt': [True, False], u'fut': [True,
False], u'prf': [True, False], u'ipf': [True, False], u'sub':
[True, False], u'tns': [True, False], u'cnd': [True, False],
u'imv': [True, False], u'cnt': [True, False]}, u'dob': {u'p2':
```

```
[True, False], u'anim': [True, False], u'xpl': [True, False],
u'fem': [True, False], u'p1': [True, False], u'fam': [True,
False], u'pl': [True, False]}, u'sb': {u'p2': [True, False],
u'p1': [True, False], u'pl': [True, False], u'fam': [True,
False]}, u'ref': {u'fem': [True, False], u'pl': [True, False],
u'xpl': [True, False]}, u'pos': [u'v', u'n', u'adv', u'adj'],
u'iob': {u'p2': [True, False], u'xpl': [True, False], u'p1':
[True, False], u'pl': [True, False], u'fam': [True, False]}}
```

4. Language-specific details

The transducers for both Spanish and Quechua work with lexicons of roots and stems; they fail on a word whose root or stem is unfamiliar. (That is, there are no "guesser" transducers in the system (Beesley & Karttunen, 2003).) The transducers have not yet been systematically evaluated for either language.

a. Spanish

The lexicon for the Spanish analyzer/generator contains approximately 3000 verb stems, most gleaned from the verbs in the Spanish WordNet.

The program handles all but a few rare irregular verbs and all of the basic orthographic changes that affect verbs. It also handles object clitic pronouns, both preverbally (where they are written as separate words) and postverbally (where they are written as suffixes). Direct objects (`dob`), indirect objects (`iob`), and reflexive pronouns (`refl`) are distinguished.

(16)
```
>>> l3.anal_word('es', 'comprármelas')
Palabra: comprármelas
CG: sustantivo, raíz: <comprar>
 infinitivo; comp dir: 3, plur, fem; reflexivo
CG: sustantivo, raíz: <comprar>
 infinitivo; comp dir: 3, plur, fem; comp indir: 1, sing

>>> l3.gen('es', 'comprar', '[iob=[+p2,+fam], dob=[-p1,-p2,+pl]]')
te los compra

>>> l3.gen('es', 'comprar', '[+refl, dob=[-p1, -p2]]')
se le compra
```

AntiMorpho is meant for the most common varieties of Latin American Spanish; thus the program does not handle subject agreement or object pronouns in the familiar second person plural (e.g., *cantáis*) of the Spanish of Spain or second person singular subject agreement in the *voseo* dialects (e.g., *cantás*).

To generate different tense-aspect-mood forms, use these boolean features within the `tam` complex feature: past (pretérito): `prt`; future: `fut`; conditional: `cnd`; imperative: `imv`; imperfect: `ipf`; present subjunctive: `sub`; imperfect subjunctive: `sub`, `prt`. Some examples:

(17)
```
>>> l3.gen('es', 'comprar', '[tam=[+prt]]')
compró
```

```
     >>> l3.gen('es', 'comprar', '[tam=[+cnd]]')
     compraría

     >>> l3.gen('es', 'comprar', '[tam=[+sub,+prt]]')
     comprara
```

The program can also analyze and generate non-finite forms of verbs: infinitives, gerunds (present participles), and past participles. To generate these forms, change the part-of-speech to noun (`n`) for infinitives, to adverb (`adv`) for gerunds, and to adjective (`adj`) for past participles. You can generate different forms of the past participle by setting the gender (`±fem`) and number (`±pl`) of the `ref` (referent) feature.

```
(18) >>> l3.gen('es', 'comprar', '[pos=adv]')
     comprando

     >>> l3.gen('es', 'comprar', '[pos=adj,ref=[+fem,+pl]]')
     compradas
```

b. QUECHUA

The lexicon behind the Quechua analyzer/generators contains 13,609 verb, noun, or adjective roots/stems and 1198 words in miscellaneous classes that are not analyzed by the program but returned along with their part-of-speech and Spanish gloss. The lexicon is extracted from the words in the Cuzco dialect in the vocabulary that is available on the RUNASIMI.DE website: http://www.runasimi.de/runaengl.htm. Some of the forms in the lexicon are genuine roots, for example, the noun root *wasi* 'house' and the verb root *muna* 'like', whereas others are stems derived from roots, for example, *wasiyuq* 'house owner', *munachi* 'cause to like'. Note that when the stem and root are both in the lexicon, there will be some spurious ambiguity in analysis.

```
(19) >>> l3.anal_word('qu', 'munachiwanki')
     Palabra: munachiwanki
     CG: verbo, raíz: <muna>
      español: anhelar, gustar, tener cariño, necesitar, querer,...
      subjeto: 2 sing
      complemento directo: 1 sing
      TAM: presente/futuro
      Derivación: causativo
     CG: verbo, raíz: <munachi>
      español: hacer querer hacer algo
      subjeto: 2 sing
      complemento directo: 1 sing
      TAM: presente/futuro
```

The basis of the Quechua morphological rules in ANTIMORFO is the grammatical accounts in Calvo Pérez (1993) and Soto Ruiz (1976). Most of the terms used for particular morphological categories are from Calvo Pérez (1993). In the discussion that follows, there is no consideration of the semantics of the various grammatical morphemes.

i. Nouns and adjectives

For the most part, nouns and adjectives are treated identically in ANTIMORFO, and they are not distinguished in the discussion that follows.

ANTIMORFO handles all of the Quechua cases. Since up to three different case suffixes are possible on a given noun, case is represented by a complex feature (`cs`) with multiple simple boolean features. The boolean features are: `acc` (accusative: *-ta*), `gen` (genitive: *-q*, *-pa*), `ben` (benefactive: *-paq*), `ins` (instrumental: *-wan*), `loc` (locative: *-pi*), `ill` (illative: *-man*), `abl` (ablative: *-manta*), `intrc` (interactive: *-pura*), `pos` (positional: *-npa*), `prol` (prolative: *-nta*), `dstr` (distributive: *-nka*), `caus` (causal: *-rayku*), and `trm` (terminative: *-kama*).

(20)
```
>>> l3.anal_word('qu', 'wasipi')
Palabra: wasipi
CG: substantivo, raíz: <wasi>
 español: vivienda, hogar, casa, posada, morada, cueva,...
 Caso: locativo

>>> l3.gen('qu', "t'anta", '[cs=[+acc,+ins]]')
t'antatawan
```

Because the noun possessive suffixes are similar to the subject suffixes on intransitive verbs and the possessive suffixes are used to represent the subjects of some nominalized verbs, they are treated under the same feature as verb subjects: `sb`.

(21)
```
>>> l3.gen('qu', 'wasi', '[sb=[+p1,+p2,+pl]]')
wasinchis
```

Note that the inclusive first person plural is represented by the feature value combination `[+p1,+p2,+pl]`.

For noun plural, use the feature `pl`.

(22)
```
>>> l3.gen('qu', 'wasi', '[+pl]')
wasikuna
```

Most of the so-called "independent" suffixes, which can appear on stems of any word class, are treated as simple (mostly boolean) features under the complex feature `prag`. These include `xplc` (explicitative: *-puni*); `nonassr` (non-assertive: *-chu*); `ev` (evidential), with values `vald` (validator: *-n/-mi*), `rept` (reportative: *-s/-si*), and `None`; `emph` (emphatic: *-á*); `resp` (responsive: *-ri*); `progn` (prognosticator: *-cha*); `dubt` (dubitative: *-sina/-suna*); `top` (topical: *-qa*); `emot` (emotive: *-y(á)*); and `intj` (interjective: *-t(á)*).

(23)
```
>>> l3.gen('qu', 'warmi', '[prag=[+nonassr]]')
warmichu

>>> l3.gen('qu', 'warmi', '[prag=[ev=vald],sb=[-p1,-p2]]')
warminmi
```

Three conjunctive suffixes are grouped under the complex feature `conj`: `add` (additive: *-pas/-pis*), `cop` (copulative: *-wan*), and `seq` (sequential: *-taq*).

(24)
```
>>> l3.gen('qu', 'runa', '[conj=[+add]]')
runapis
```

Most derivational morphemes are represented as boolean features under the complex feature `der`. These include noun-to-noun, noun-to-verb, and verb-to-verb suffixes. The noun-to-noun features are `dim` (diminutive: -*cha*), `aug` (augmentative: -*sapa*), `pos` (possessive: -*(ni)yoq*), `incl` (inclusive: -*ntin*, also verb-to-verb), and `lim` (limitative: -*lla*, also verb-to-verb).

(25)
```
>>> l3.gen('qu', 'warmi', '[der=[+dim],conj=[+cop]]')
warmichawan
```

Nominalized forms (verb-to-noun derivation) are handled with the part-of-speech feature `pos2`, which has as possible values `inf` (infinitive: -*y*), `agt` (agent: -*q*), `nft` (future nominal: -*na*), `prt` (participial: -*sqa*), `ger` (gerund: -*spa*), and `sub` (subordinate: -*qti*).

(26)
```
>>> l3.gen('qu', 'muna', '[pos2=ger]')
munaspa

>>> l3.gen('qu', 'muna', '[pos2=sub,sb=[+p2]]')
munaqtiyki
```

ii. Verbs

Like nouns and adjectives, verbs can take suffixes within the `prag` and `conj` features.

Verb tense, aspect, and mood are represented under two complex features, `tam` and `asp`. `Tam` features include `ft` (future: -*sa*/-*qa*/*0*); `ps` (past: -*r(q)a*); `pp` (past perfect: -*sqa*), `cd` (conditional: -*man*/*0*), `im` (imperative: -*y*/-*chu*). `Asp` features include `prg` (progressive: -*sha*); `contin` (continuous, `y`: -*raq*; `n`: -*ña*; `None`). When all `tam` features are `False`, the verb is in the present tense.

(27)
```
>>> l3.gen('qu', 'riqsi', '[pos=v,tam=[+ft]]')
riqsinqa

>>> l3.gen('qu', 'riqsi', '[pos=v,tam=[+cd],ob=[+p2]]')
riqsisunkiman

>>> l3.gen('qu', 'riqsi', '[pos=v,asp=[+prg],sb=[+p1]]')
riqsishani
```

Only the verb-to-verb derivational morphemes that appear to be the most productive are handled by ANTIMORFO. The boolean features for these morphemes are `rfl` (reflexive: -*ku*); `rcp` (reciprocal: -*naku*); `des` (desiderative: -*naya*); `cisloc` (cislocative: -*mu*); `caus` (causative: -*chi*); `cnsc` (consecutive: -*sti*); `lim` (limitative: -*lla*), and `incl` (inclusive: -*ntin*). Two noun-to-verb (or adjective-to-verb) morphemes are also covered: `fct` (factitive: -*cha*) and `inch` (inchoative: -*ya*).

(28)
```
>>> l3.gen('qu', 'riqsi', '[pos=v,der=[+rcp],sb=[+pl]]')
riqsinakunku

>>> l3.gen('qu', 'amuq', '[pos=v,der=[+inch],tam=[+ps],sb=[+p2]]')
amuqyarqanki
```

References

Beesley, K.R., and Karttunen, L. (2003). *Finite state morphology*. Stanford, California: CSLI Publications.

Calvo Pérez, J. (1993). *Pragmática y gramática del quechua cuzqueño*. Cuzco, Peru: Centro de Estudios Regionales Andinos Bartolomé de las Casas.

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*.

Soto Ruiz, C. (1976). *Gramática quechua: Ayacucho-Chanca.* Lima, Peru: Ministerio de Educación/Instituto de Estudios Peruanos.