SOFTWARE REVIEW -- ALCHEMIST

Stephen Grimes
stgrimes@indiana.edu
Indiana University

INTRODUCTION

Alchemist is a tool for morphological annotation that was developed by Colin Sprague and Yu Hu at the University of Chicago as part of the Linguistica Project supervised by John Goldsmith. The overall aim of the Linguistica Project is to develop a set of heuristics that can produce an unsupervised morphological analysis of a language. Within this larger framework, the primary function of Alchemist is to aid in developing a gold standard in order to evaluate the results of a morphological analysis. Alchemist is intended to be used by a linguist who is already familiar with the morphology of the test language under consideration. In this review, I am additionally interested in whether Alchemist is of more general use to field linguists or morphologists who are not directly interested in unsupervised learning of morphology.

PROGRAM OVERVIEW

At program initialization, the linguist begins by selecting the input to be used -- this can either be a text corpus or a dictionary (a tokenized list of words). In general, using a running text corpus is advantageous to using word lists, as a corpus includes distributional and frequency data; furthermore a corpus contains both derived and inflected morphological forms, while a dictionary might not contain inflected forms. Of course the quality of the input is paramount, as a text that is not monolingual or not "cleaned" of stray characters or punctuation is not very useful.

After basic setup in Alchemist is completed, the linguist begins the arduous task of analyzing words and identifying morphemes. Each morpheme has a set of predefined attributes to be specified (user-defined attributes are also possible). The only attribute that is required to be specified is whether a morpheme is a root or an affix. One of the more interesting program features is that each morpheme has a confidence attribute, corresponding to a linguist's belief in how "good" a morpheme it is of the language. This concept is not defined, but rather it corresponds to the linguist's "intuition", which is likely advised by frequency, semantic, morphophonological, or morphological productivity considerations. So, for example, the English verbal suffix -ing would receive a high confidence rating, while English plural -en (as in 'children' or 'oxen') would receive a low rating, if it is to be considered a morpheme at all. By having a stratified hierarchy for the morphemes in a language, this establishes a gradient confidence scale that an unsupervised morphological parser would attempt to replicate.

Unfortunately, each word must be analyzed individually, as there is no ability to parse unseen words based on the analyses of previous words. For a large corpus, this annotation process would seem to be very labor-intensive and may involve repeatedly identifying the same morpheme. When the morphological analysis is complete, an XML file is generated using <word> and <morpheme> tags, as shown below for the English word "singing". In the XML representation below, "singing" has been segmented into two morphemes, and the morpheme boundary is deduced by the values for the "start" attribute.

```
<word comment="" morphemes="2" key="singing" index="1" >
  <morpheme comment="" morphemeindex="7" type="1" score="1"
    start="0"   color="4"/>
 <morpheme comment="" morphemeindex="51" type="2" score="1"
    start="4" color="5"/>
</word>
```

Alchemist can also export a morphologically annotated corpus as table-formatted text in order to be more easily readable for humans. As such, Alchemist serves as a tool for developing a morphologically-based corpus for a language, which may serve as an end in itself, or be a first step in a line of further research.

PROGRAM FEATURES

Alchemist makes use of regular expressions (pattern matching) in different modules of the program. The developers feel that this capability may be a reason in itself to use Alchemist -- the user has the ability to search the lexicon for any word that fits a given pattern. At initialization, Alchemist can "scrub" an input text using built-in or user-defined regular expressions in order to eliminate unwanted characters, such as email headers. Examples of built-in regular expressions include the ability to convert all letters to lowercase, remove digits, and remove certain leading and trailing punctuation symbols. The regular expression syntax should be familiar to someone accustomed to Perl, but the graphical user interface (GUI) in Alchemist in theory makes using these regular expressions less intimidating.

The lexicon can be viewed in one of the three main windows of the program in a number of different sort orders. The typical ordering is alphabetical or reverse alphabetical (alphabetized from right-to-left instead of left-to-right). The user can choose to only view analyzed or unanalyzed words. Filters allow other subsets of the lexicon to be viewed using regular expressions, so one can look at words ending only in -ing (using the regular expression /ing$/), or one can find all words with no high vowels (/^[^iu]*$/).

All morphemes have one required attribute: they must be designated as either an affix or a root. (Note that for compound words, the ability to have multiple morphemes labeled as roots becomes useful). Attributes that can be optionally specified are :

Type (Root, Affix, Suffix, Prefix, Infix, Other)
Part of speech (Adjective, Noun, Proper noun, Verb, Other)
Tense (Present, Past simple, Past imperfect, Subjunctive, Other)
Polarity (Affirmative, Negative)
Person (1st, 2nd, 3rd, Other)
Class (Class numbers through 20 are given, Other)
Number (Singular, Dual, Plural, Other)
Gender (Feminine, Masculine, Neutral, Other)

Clearly no one morpheme would have all of the above attributes specified, as some of the above are more useful for nouns or for verbs. User-defined attributes are also allowed in order to

classify more complex morphologies.

Multiple instances of a single word may be defined in the case that one orthographic form would have multiple morphological analyses. An example given in the program documentation is an orthographic form like “polish”. It has both adjectival and verbal uses: pol+ish (something from Poland) vs. polish (verb). (Whether one would want to distinguish between “can” (to be able) and “can” (a container) would be a related but secondary issue. In this version of Alchemist, semantics only exists to the extent that words are defined by their attribute categories.)

The best form of user help is not the help menu in the program, but a .pdf file downloadable from the Alchemist website. The help describes some features in more detail than others and some aspects of the program are left somewhat undocumented, but overall it is a necessary read and is generally quite useful. A special form of help available within the Alchemist itself is used by first clicking a "question mark" icon, which changes the program state so that the next immediate click is treated as a help query on whatever item is clicked, although this function had limited implementation.

EVALUATION

Overall the program is quite user-friendly. Alchemist is not overloaded with menu options, and hence program features are easy to find and there is a fairly quick learning curve. The judicious use of color for morpheme highlighting is also visually quite appealing.

Unfortunately, I managed to find ways to “crash” the program. Some instances of critical failure occurred while I was arbitrarily clicking in different windows in order to see what functionality was available. Other crashes seemed to occur while working with the merge morpheme function. As mentioned above, separate instances of homographic morphemes are initially treated as distinct morphemes. This default is advantageous when a language has many morphemes that sound identical, such as -s in English (which is used as a plural, possessive, and third person singular marker). However, this is a burden when one must explicitly merge all instances of a morpheme whose phonetic form is unambiguous, such as English -tion. The merge function changes the internal storage structure of the morphological lexicon, and I suspect program crashes may be due to corrupted internal storage.

Additionally, for me it was not clear how to "unmerge" two morphemes once they have been merged together. One solution is to delete the word from the database, then create a new entry to replace it, but I thought there should be a more straightforward way of doing this (and perhaps there is, although it is not readily apparent).

A number of linguistic limitations of Alchemist arise from a restricted definition of what a morpheme is allowed to be. Here morphology is taken to be morphology without underlying forms, and paradigms only exist to the extent that words share common substrings. Due to the nature of the representations, a morpheme must consist of a positive integer number of segments/phonemes. Given the hypothetical underlying form /mi/ + /ig/ realized as [mig], the [i] segment is shared by two morphemes. In Alchemist the possibilities for abstraction are limited -- neither /mi/ + /g/ nor /m/ + /ig/ would be ideal segmentations, but here one is forced to make a

decision. (Perhaps this scenario is a candidate to use Alchemist confidence level to score these particular morphemes lower, but this is not ideal, as in general there is no clear relationship between morphological productivity and phonological opaqueness.)

Similarly, a morphological process such as umlaut, circumfiction, or one which refers directly to a non-segment or suprasegmental, such as for instance in moraic morphology, will be problematic in Alchemist. Ultimately the linguist is responsible for determining what to take as the privatives in the language (the alphabet), and some creative changes to basic representation may alleviate problems. For example, in a tonal language where the morphology is sensitive to whether there is a high or low tone on the vowel, one might double the number of vowels to create a high tone and low tone instance of each vowel in order to represent this morphology in Alchemist. In general, however, without allowing for multiple-dependencies between single surface segments and non-segmentals, it seems challenging to establish a reasonable "morphological gold-standard" for most languages, except for perhaps Chinese.

Overall, however, I see Alchemist as an excellent development, as there appears to be a significant lack of morphologically annotated corpora that are publicly available. The possible uses of such corpora are not simply limited to morphological learning, however. Just as computational resources have allowed phonotactics to become viewed as probabilistic, making tools available to create morphologically annotated corpora more widely available will allow further developments in probabilistic morphotactics. Similarly, bilingual parallel corpora with morphological annotation can lead to developments in machine translation. Whether the regular expression capabilities of Alchemist will ultimately be used by less tech-savvy linguists remains to be seen, but this aspect of the program could potentially be used by phonologists to find crucial examples of words of a given phonological form. I challenge and encourage anyone interested to consider the features of the program in order to find potential applications in divergent fields of linguistics.

The program is available for downloading at:
http://linguistica.uchicago.edu/alchemist.html

BIOGRAPHY

Stephen Grimes is a Ph.D. student in the Linguistics Department at Indiana University and is interested in phonology, morphology, computational linguistics, and Hungarian linguistics. He currently holds a position the American Indian Studies Research Institute at Indiana University, working to support development of the Automated Text Processor, a utility to aid field linguists in dictionary development and morphological analysis of texts. He is also working on examining unsupervised methods for morpheme identification in polysynthetic and agglutinative languages. His personal webpage is located at <http://mypage.iu.edu/~stgrimes>.