Morphological parser for Latin

Alexander Malmberg
LTH
d00am@efd.lth.se

Abstract

Morphology describes how words are formed in a language, for example by adding suffixes or prefixes to existing words. In some languages, this process is very productive, and it is thus important for computational linguistics to be able to handle this. The purpose of a morphological parser is to extract information from the morphological structure of a word. In this paper, we examine this problem and briefly look at the standard two-level morphology approach of handling it. We also present a basic but working morphological parser for Latin.

1 Introduction

Morphology is the study of how words are formed. In many languages, the processes by which new words are formed are very common. For example, in English, one can form compound words, and it is common that plural forms of words are formed by adding "s" to the singular form. Other languages use other sets of prefixes and suffixes to form new words from other words, sometimes with phonological changes (such as "morphologies", where "ys" turns into "ies"). Some languages use infixes or other exotic methods for forming words.

Systems that want to process text in a language need to understand all these words. A simple and straightforward approach is to make a dictionary that lists all words. However, this is ugly from a theoretic point of view. Many of the methods that form new words are regular, and it should be possible to build a model of these methods and use it.

It is also impractical to list all words, especially in languages with rich morphological processes. For example, nearly every Latin verb has approximately 150 forms, but these can usually be formed from just three stems. Even in English, which has relatively poor morphological processes, listing all words is unlikely to work in practice. An interesting example(Sproat, 1992) involved Associated Press newswire text from a 10 month period. Even when the words from all the texts expect those of the last day of the period were collected in a dictionary, there were still many words on the final day that weren't in the dictionary. Many of these involved new forms of words that were in the dictionaries.

Thus, morphology aims at modelling how words are formed, and the job of a morphological parser is to extract information from words using this model. There are many applications of this, and different applications need different types of information. One type would be information about gender, number, tense, etc., which could be used to find the meaning of a word, or to aid part-of-speech tagging. Other applications include spell checking, or text-to-speech, where morphology can provide information about morpheme boundaries and pronunciation.

2 Two-level morphology

One standard way of writing a morphological parser is to use so called two-level morphology. This was originally done by Koskenniemi in the KIMMO system for Finnish.

The first level in two-level morphology is, roughly, a "dictionary" with idealized morphological rules. The second level is a set of phonological rules for rewriting idealized forms of words into their real forms.

The "dictionary" can be represented as a web of tries. A trie is a tree where each node has a child for each letter. This makes it possible to find the node for a word effi-

ciently: just start at the root and recursively go to the child corresponding to the next letter. Morphological rules are handled by connecting many tries; the node for a stem won't have children for all the possible endings. Instead, it will have a link to a separate trie that contains these endings. This way, only one trie is needed for each (idealized) paradigm, and it is still possible to find the node for a complete word efficiently.

The phonological rules are represented as finite state automatons that accept or reject a pair of strings. One of the strings would be a real form of a word, and the other would be an idealized form as found in the dictionary. The automaton would accept the pair if the dictionary form matches the real form.

When parsing words, these two levels run in parallel. The dictionary trie is searched recursively starting at the root. At each node, the idealized form (so far) is compared to the real form using the automaton to see if the idealized form might correspond to the real form. If it doesn't, the search need not continue below that node. (Since the phonological rules might include large changes, the system might have to search a few levels down dead-ends before the automaton can reject the pair.)

There are many practical details in implementing such a system, but this is only a brief description. A more extensive description can be found in my source for this section (Sproat, 1992).

3 Morphology in Latin

Morphology in Latin is extensive: nearly every word indicates number and gender, there are many cases, many paradigms, and many obscure forms of verbs.

However, the structure is fairly simple: words are formed by adding suffixes to a stem. There are no phonological rules (except some vowel length changes, but since I'm working with written texts, that doesn't affect my parser). Completely new words are can be formed using prefixes, but these were included in my dictionary and thus didn't cause any problems. Stems are formed in more complex ways, but again, listing all stems isn't hard (e.g. a verb may need 3–4 stems, but no more).

(It is perhaps worth noting that classic Latin is a language where it would be possible to simply build a list of all words. Being a dead language, no new texts will be written in it, so if you collected all words in all texts, you'd trivially get perfect coverage on all texts.)

During the work on my parser, I used primarily two Latin grammar references: Grammatica Nova (Larsson and Plith, 1992), and Latin Grammar (Conrad, 2004).

4 Morphological parser for Latin

I wrote a morphological for Latin. It is based on the dictionary level of the two-level morphology and doesn't include any phonological rules. The trie structure is defined in trie.h and the main source is in latin1.c.

When the program is, it reads the data files specified on the command line. Each data file defines a trie: the words contained in it, which other tries it links to, and some other interesting information (e.g. the meaning of a stem, and tense/case/etc. information for endings).

The parser function is parse(). The parse is done in three steps. First, the tries are searched for the unmodified word.

If no parses are found and the word ends in "que", the "que" is removed and the search is attempted again. "que" is a word that is sometimes attached to other words as a suffix. Since it can be attached to all kinds of words, it was convenient to special case this word here.

If no parses are found in the second search, the parser tries to parse the word as a roman numeral.

Writing and debugging the parser was fairly easy. Most of the time in the project was spent gathering and working on the data that the parser uses.

4.1 Data

When the parser is run, it is given a list of files with data that is used to build and link together the tries. There are two basic kinds of tries: stems and endings.

4.1.1 Endings

The ending tries were built by hand by me using Latin grammar resources (Larsson and

Plith, 1992) (Conrad, 2004). While it would have been possible (and straightforward) to simply make long lists of all endings from a grammar book, there are many regularities in the endings, and I tried to exploit this.

As an example, almost all verb forms use one of three sets of endings to indicate person. Thus, instead of having to list 6 endings for each combination of verb conjugation, tense, active/passive, etc., only the first part of the ending is listed along with a link to the trie with person endings corresponding to this combination. (In fact, in some cases I cheat and do this even when some forms don't follow one of the three patterns. In those cases, I also list the exceptional forms, so the parser still recognizes all valid forms; the drawback is that it will also recognize some ill-formed words.)

With some support for handling phonological rules, it would have been possible to exploit even more near-regularities. Unfortunately, the near-regular endings don't seem to follow regular phonological rules. For example, the ending for both nominative plural and genitive singular second declension nouns is "-i". For second declension nouns whose stems end in "i", such as "gladius", the "ii" in genitive singular is contracted to a single "i", "gladi", while the "ii" in nominative plural isn't, "gladii".

To handle this in a two-level morphology, it would have been necessary to introduce new "magic" letters, e.g. several variants of "i", identical except that some would combine with other "i":s and some wouldn't. Thus, you still wouldn't really be able to exploit the regularities since you'd have to explicitly list which "i" would be used in different endings. To me, this doesn't appear to be any nicer than simply listing all endings from a theoretical point of view.

4.1.2 Stems

The stems were collected from a dictionary built from the word list of another morphological parser for Latin (Whitaker, 2004). This dictionary included over 30000 entries, and while it was written in traditional dictionary form, it included enough information about the words to extract the stems and connect them to my ending tries.

The program gen_roots_dict_1 parses the dictionary and builds the data files used by my parser. The program can handle about 25000 of the entries in the dictionary. Extending the coverage is straightforward but, at this stage, time consuming since the remaining words are spread across many small paradigms.

Author	Number of words	Coverage
Caesar	51624	91%
Vergilius	63748	77%

Table 1: Parser results

5 Results

I tested my parser on "Commentariorum Libri VII de Bello Gallico" by Caesar, and the Aeneid by Vergilius (both texts from "Corpus Scriptorum Latinorum" (Camden, 2003)). The results are in table 1.

While developing the parser, I tested and analyzed the results of the parser on parts of the first chapter of the text by Caesar. These results were used to guide the development; they told me which words and paradigms that would increase coverage the most. Since the results from the Caesar text was used for this, this is likely part of the reason why the coverage is much better on the Caesar text. Another reason could be that the dictionary I extracted stems from was, according to its documentation, originally built using Caesar's texts.

On average, there were about 2.5 parses of each successfully parsed word. Many of these seem to be cases where several different genders/cases of a word have the same ending.

I have examined correctness by manually examining some randomly selected words, and by systematically testing some paradigms, and to the limits of my knowledge of Latin, all parses are valid (and usually, if a word is parsed at all, the correct parse is among the parses found).

References

David Camden. 2003. Corpus scriptorum latinorum, http://www.forumromanum.org/literature/index.htm

Eric Conrad. 2004. Latin grammar, http://www.math.ohio-state.edu/ econrad/lang/latin.html.

Lars A. Larsson and Håkan Plith. 1992. *Grammatica Nova*. Bonniers.

Richard Sproat. 1992. *Morphology and Computation*. ACL-MIT Press.

William Whitaker. 2004. Words 1.97, http://users.erols.com/whitaker/.