Constructing a Parser for Latin

C.H.A. Koster

Computing Science Institute,
University of Nijmegen,
The Netherlands
kees@cs.kun.nl

Abstract. We describe the construction of a grammar and lexicon for Latin in the AGFL formalism, in particular the generation of the lexicon by means of transduction and the description of the syntax using the Free Word Order operator. From these two components, an efficient Top-Down chart parser is generated automatically. We measure the lexical and syntactical coverage of the parser and describe how to increase it.

The morphological generation technique described here is applicable to many highly-inflected languages. Since the Free Word Order operator described can cope with the extremely free word order in Latin, it may well be used for the description of free-word-order phenomena in modern languages.

1 Introduction

Why would anybody in his right mind construct a formal grammar of Latin? Although there exist some active speakers of the language and according to some its best poetry was produced in the nineteenth century, the language is as dead as a doornail. A large corpus of latin texts is extant, but there is no expectation of any important additions. Most texts have been translated into many other languages in which they can be enjoyed without the drudge of learning Latin. Commercial application of an Information Retrieval system for Latin is inconceivable. Furthermore there already exists an overwhelming number of learned grammars and dictionaries for it, to which any formal grammar would add nothing new.

It was for the Latin language, and earlier for Greek, that the science of linguistics as we know it was developed. Everyday concepts and terminology of Latin still pervade western linguistic thinking. The understanding of the structure of Latin provides a framework in which not only its linguistic relatives, but also utterly unrelated languages could be analysed, modeled and described. The Latin language has a number of properties (detailed and rich morphology, very free word order) which together with its quite regular structure make it an interesting object for formal description. Practically, it is the mother of the Romance languages and the aunt of many other languages (including English) which do have practical and even commercial value. Lastly, describing it with the aid of modern grammar and parsing technology may be of therapeutic value for one who has been forcefed on it for a number of years in high school.

A. Gelbukh (Ed.): CICLing 2005, LNCS 3406, pp. 48–59, 2005.

1.1 About Grammars

Two utterly differents kind of grammars can be distinguished:

1. **Grammar$_1$**: to real linguists, a grammar is a thick book, in which every aspect of the structure of a certain language is described in an informal but highly rigorous fashion.
2. **Grammar$_2$**: to computer scientists and computer linguists a grammar is a description of the morphosyntax of a language in a formalism such that (given also a lexicon) a parser can be constructed from it by automatic means.

Of course there are incredibly many variations on these concepts, but the basic fact is that any linguists happily working with the one kind of grammar has very little patience for the other kind.

In this paper we try to reconciliate the two, deriving a practically functioning grammar$_2$ (a *formal* grammar) from the knowledge contained in a grammar$_1$. We base ourselves on [Redde Rationem] and [Latijnse Leergang].

1.2 About Latin

The Latin language presents a challenge to its description in current syntactic formalisms and automatic parsing methods, due to the fact that it is definitely not a Context-Free language:

- it has a rich morphology and agreement rules, governed by (classical) features like Number, Person, Gender, Case, Time, Voice and Tense.
- it displays a close approximation to Free Word Order, in that nearly all constituents of a phrase can be permuted without affecting the meaning of the phrase.

In fact, the family of two-level grammars was conceived for the formal description of such rich morphological and agreement rules (although mostly motivated by the description of programming languages with their typing and identification rules, rather than natural languages).

In this note, we shall make use of AGFL [Koster, 1991], which is a member of the family of two-level grammars.

1.3 About AGFL

Affix Grammars over a Finite Lattice (AGFL) are a form of two-level grammars in which the features take on as values any subset of a given finite set. The typical examples of such features are completely classical: an affix `NUMBER` distinguishing between singular and plural, and another affix `PERSON` distinguishing between the first, second and third person may be defined in AGFL by the affix rules

```
NUMBER :: sing | plur.
PERSON :: first | secnd | third.
```

An affix may take on a set-value (any non-empty subset of its domain) rather than a specific single value, indicating partial knowledge, e.g. `PERSON = {first | third}` indicates the knowledge that the Person-feature does not have the value `second`.

Affixes are used as parameters to the nonterminal-rules of the grammar to indicate agreement, e.g.

```
sentence:
   subject(NUMBER,PERSON), verb(NUMBER,PERSON), object.
```

where the *consistent substitution rule* ensures that different occurrences of the same affix must have the same value (enforcing agreement in Person and Number between subject and verb). The notation of AGFL should be similar enough to PROLOG with DCG's (which is indeed a related formalism) to allow a computer linguist to comprehend the following examples without further explanation.

An AGFL grammar describes the constituency structure of a language, with the commas in a syntax rule indicating sequential order and the semicolons indicating alteration. However, there is also a proviso for Free Word Order (FWO): members separated by ampersands my occur in any order. The development of a grammar for Latin provided a nice opportunity to exercise (and debug) this facility, which is intended for the compact description of FWO languages.

AGFL also allows a rule to describe a (compositional) *transduction*: every alternative may indicate how its translation is to be composed out of the translation of its members (where a terminal symbol is translated to itself). We'll use this transduction instead of parse trees to show the results of parsing, but also use it to construct our lexicon.

2 Constructing a Latin Lexicon

Although some lists of Latin words are freely available on the internet, there is no machine-readable lexicon to be found, and we had to develop one from scratch.

In principle, the word(forms) of a language together with their parts of speech (POS) may simply be enumerated in the grammar by rules like

```
LEXV(ind,prm,sg,perfct,act): "amavi".
```

but there would be very many of such rules, and the efficiency of recognizing the enumerated wordforms would be terrible. A more principled approach would be to enumerate a list of stems and lists of infixes and suffixes according to their conjugation, combining them by rules like

```
VERB(ind,prm,sg,perfct,act):
  V_STEM(CONJ,act),
    V_INFIX(CONJ,perfct),
      V_SUFFIX(perfct,prm,sg,act).
```

It is quite possible to enumerate the stems and fixes in the lexicon, so that they do not clotter up the grammar and are recognized by means of lexicon lookup rather than by exhaustive trial. Lexicon entries (in the notation of AGFL) look like

```
"ama-"  V_STEM(a_conj,act)
"-v-"   V_INFIX(a_conj,perfct)
"-i"    V_SUFFIX(perfct,prm,sg,act)
```

While this approach is feasible, it takes a lot of effort describing the morphology of Latin in great detail (copying all this wisdom from a grammar$_1$). Especially the irregular or partly regular words will lead to many fine distinctions. It therefore makes sense to generate those irregular forms once-and-for-all and put the complete wordforms into the lexicon:

```
"sum"   TOBE(ind,prm,sg,praes,act)
"es"    TOBE(ind,sec,sg,praes,act)
"est"   TOBE(ind,trt,sg,praes,act)
"sumus" TOBE(ind,prm,pl,praes,act)
"estis" TOBE(ind,sec,pl,praes,act)
"sunt"  TOBE(ind,trt,pl,praes,act)
```

However this idea points to another approach which is more uniform and simple: to generate also *all* regular wordforms once-and-for-all and put them into the lexicon. Rather than recognizing a wordform from its parts the parser will then recognize a wordform as a whole, by means of whole-word lexicon lookup. A `generative` solution, rather than an *analytic* one.

An objection to this approach may be seen in its efficiency – since very many wordforms may come form one stem, we would have a very large lexicon. Indeed a simple verb like contestare will generate 168 wordforms, including the declinated forms of participles and even some adverbs. The lexicon system of AGFL (using compacted trie structures) is however highly efficient, both in time (faster than parsing the parts) and space (the lexicon takes about the same space as the list of all words contained in it), so that the efficiency is actually better than in the previous solution.

Another objection might be the severe overgeneration expected from putting all forms in the lexicon, without verifying whether they are attested in any text. However, the same objection applies to the corresponding analytical approach which would recognise precisely the same wordforms. Word forms which do not "exist" will not be used, it is as simple as that. But it should be noted that, in order to reach sufficient coverage of the lexicon in spite of limited effort, certain rules will have to be included in the grammar in order to "guess" the POS of out-of-vocabulary words; thus the overgeneration can be seen as a positive contribution to robustness! Anyway, the overgeneration can if needed be avoided by "filtering" the lexicon through a wordlist obtained from a large corpus.

This then is the approach we have taken: to generate the lexicon for the large open classes (noun, verb, adjective and adverb) from a very classical resource:

word lists and stem times, as contained in any standard grammar$_1$C, with a separate treatment for irregular words (that can be generated in the same way, correcting the irregularities by hand).

2.1 Metarules for the Lexicon

The metarules defining the affixes used in the lexicon with their domains are the following:

```
CASUS:: nom | voc | gen | dat | acc | abl | loc.
```

The six cases of Latin.

```
NUM:: sg | pl.
GENUS:: fem | masc | ntr.
PERS:: prm | sec | trt.
```

Now come the affixes detailing the POS of verbs:

```
MODUS:: ind | con | imp | inf | part | pperf | gerund.
TEMPUS:: praes | imprf | futur | perfct | pqperf | futex.
VGENUS:: act | pas.
```

The following affix pertains to adjectives and adverbs, which are also generated from verbs:

```
GRADUS:: pos | comp | super.
```

2.2 Nouns

The noun entries are generated from a list of entries, one per line, for different declinations, like:

```
a poeta masc
a plaga fem
b poculum
b populus
c portio portionis fem
c plebs plebis masc
e facies masc
i aer aeris masc
ie hostis hostis masc
```

From these list entries, lexicon entries are generated of type

```
 LEXS(NUM,GENUS,CASUS)
```

by means of a grammar describing the transduction from a list entry to all corresponding lexicon entries. The following rule for the a-declination is typical:

```
declinatio prima:
   "a", radix, "a", ",genus." /
     "\"",radix,"a\"\tLEXS(sg,",genus,",nom|voc|abl)\n",
     "\"",radix,"ae\"\tLEXS(sg,",genus,",gen|dat)\n",
     "\"",radix,"am\"\tLEXS(sg,",genus,",acc)\n",
     "\"",radix,"ae\"\tLEXS(pl,",genus,",nom|voc)\n",
     "\"",radix,"arum\"\tLEXS(pl,",genus,",gen)\n",
     "\"",radix,"is\"\tLEXS(pl,",genus,",dat|abl)\n",
     "\"",radix,"as\"\tLEXS(pl,",genus,",acc)\n".

genus: "masc"; "fem"; "ntr".

radix:$MATCH(".*-").
```

Notice that the root of the word is matched by a Regular Expression. For the word puella (feminine) the transducer generates

```
"puella"        LEXS(sg,fem,nom|voc|abl)
"puellae"       LEXS(sg,fem,gen|dat)
"puellam"       LEXS(sg,fem,acc)
"puellae"       LEXS(pl,fem,nom|voc)
"puellarum"     LEXS(pl,fem,gen)
"puellis"       LEXS(pl,fem,dat|abl)
"puellas"       LEXS(pl,fem,acc)
```

The list of regular nouns is 1429 entries long, generating 36798 lexicon entries, which are extended with 28 irregular forms.

2.3 Other Categories

Verbs also come in different conjugations; as in traditional grammars, a verb is specified by giving its infinitive, perfectum and participium perfectum or futurum, of which the latter two may be missing (dash):

```
a obscurare obscuravi obscuratus
c abhorrere abhorrui -
e abstinere abstinui abstentus
io abicere abieci abiectus
i adoriri - adoriturus
```

Note that deponentia are indicated by a passive infinitive. The output of the transduction process is a list of lexicon entries of the following types:

```
  LEXV(MODUS,PERS,NUM,TEMPUS,VGENUS)
  LEXV(inf,TEMPUS,VGENUS)
  LEXV(MODUS,NUM,GENUS,CASUS)
  LEXA(GRADUS,NUM,GENUS,CASUS),
  LEXX(GRADUS)
```

(V stands for verb, A for adjective and X for adverb). Some examples of each (for the verb amare):

```
"amo"   LEXV(ind,prm,sg,praes,act)
"amabatis"  LEXV(ind,sec,pl,imprf,act)
"amasti"    LEXV(ind,sec,sg,perfct,act)
"amare" LEXV(inf,praes,act)
"amari" LEXV(inf,praes,pas)
"amando"    LEXV(gerund,sg,masc|ntr,dat|abl)
"amantes"   LEXV(part,pl,masc|fem,nom|voc|acc)
"amaturi"   LEXA(pos,pl,masc,nom|voc)
"amabilis"  LEXA(pos,sg,GENUS,nom|gen|voc)
"amabiliter"    LEXX(pos)
"amatius"   LEXX(comp)
```

Verbs are much more productive than nouns, and there are about as many verbs as nouns, so that it is no wonder that they generate most of the lexicon entries: The 1242 entries in the verb list generate 179404 lexicon entries, to which 4339 irregular verb forms are added.

Finally, there is a list of adjectives of three different declinations: The 624 different adjective entries generate a total of 71718 lexicon entries, to which 122 irregular forms have been added by hand.

The remaining (closed) lexical categories include numerals, adverbia and vasrious kinds of pronomina. Of these lexical types there are a total of 1606 lexicon entries.

2.4 Achieving Coverage

The first word lists were created from the lists and examples given in the two textbooks. Then a grammar `sweep.gra` was developed for analysing the coverage of the lexicon. It transduces every word of the input text to a copy marked with a rough lexical category, according to the schema:

```
sententia:
   known word / marker, known word;
   looks like name / !
   any word / "?:", any word.
```

using the following markers for known words:

`V:` verb form, form of esse
`N:` noun form
`A:` adjective
`Q:` quantity
`X:` adverbium
`P:` pre- or postposition
`D:` determiner, demonstrative or pronoun

Unknown words are marked by ?:, potential names (unknown words starting with a capital letter) are skipped.

A text corpus (St. Augustine's Confessiones) was cut into words and swept. A frequency list was built from the marked words, using standard UNIX utilities

```
cat corpus| tr -cs "[:alpha:]" "[\n*]"| sweep -tP1| grep ':' >words
sort +1 words | uniq -c | sort -nr > freq
```

Then some days were spent analysing the unrecognized wordforms, from the highest frequency downwards, then extending the word lists and bootstrapping, until I got bored. At that point the lexical coverage (number of known words in the corpus divided by the total number of words) was 92%. Applying the same lexicon to the Vulgate translation of the Psalms gave a coverage of 87%.

It is striking to see the low level of lexical ambiguity of Latin compared to English, once a few cases of systematic overlap (e.g. participia and adjectiva) are resolved. The general strategy is to remove all nouns and adjectives from the word lists that can be generated from a verb.

3 Constructing the Grammar

We shall describe the grammar of Latin in a Bottom-Up manner, which was also roughly the order in which it was constructed and tested.

The lexical interface on which it rests has already been described in the previous section. The basic approach is

- describe the noun phrase as a noun generalized by projection, and similarly the adjective phrase as a generalized adjective and the verb phrase as a generalized verb form (which is optional, implying TOBE) together with its complements
- for every composed phrase enumerate the possible orders of its constituents
- then describe the way in which sentences can be glued together into longer sentences.

This is a general strategy which works reasonably well for many languages; but of course it encounters many problems and complications.

3.1 The NP

We describe the derivation of the Noun Phrase syntax in some detail. The noun phrase has as its kernel an N (standing for nomen), which may have several realizations, among which a lexical noun is preferred. An adjective phrase (AP) or a quantity is also accepted.

```
N(NUM,GENUS,CASUS):
   LEXS(NUM,GENUS,CASUS);
   $PENALTY,robust nomen(NUM,GENUS,CASUS);
   $PENALTY,AP(pos,NUM,GENUS,CASUS);
   $PENALTY(2),LEXQ(NUM,GENUS,CASUS).
```

In order to achieve some lexical robustness, there are rules for guessing the type of out-of-vocabulary words. These are only invoked for words which have no lexicon entry.

```
Nbar(CASUS),Nbar(NUM,GENUS,CASUS):
   N(NUM,GENUS,CASUS);
   $PENALTY, PRDET(NUM,GENUS,CASUS);
   PRPER(PERS,NUM,CASUS);
   $PENALTY, PRDEM(NUM,GENUS,CASUS);
   AP(pos,NUM,GENUS,CASUS), Nbar(NUM,GENUS,CASUS);
   PRPOS(NUM,GENUS,CASUS), Nbar(NUM,GENUS,CASUS);
   N(NUM,GENUS,CASUS), AP(pos,NUM,GENUS,CASUS);
   N(NUM,GENUS,CASUS), PRPOS(NUM,GENUS,CASUS).
```

Besides an N also certain pronouns are accepted, and an AP or possessive pronoun is admitted as a modifier. Note that these modifiers may precede or follow the N. The description in the last four lines is not as symmetric and general as we would like, it would be better to have

```
   Nbar(NUM,GENUS,CASUS), AP(pos,NUM,GENUS,CASUS);
   Nbar(NUM,GENUS,CASUS), PRPOS(NUM,GENUS,CASUS).
```

but the AGFL system presently does not allow left-recursion.

At this level also the clitics -que and -ve are introduced:

```
NBAR(pl,GENUS,CASUS):
   N(NUM,GENUS,CASUS), N(NUM1,GENUS,CASUS),clitic.
```

One level higher, the numerals are introduced, as well as the relative sentence, the genitive adject and explicative interjections:

```
 Nbarbar(CASUS), Nbarbar(NUM,GENUS,CASUS):
    numerus(NUM,GENUS,CASUS),Nbar(NUM,GENUS,CASUS);
    Nbar(NUM,GENUS,CASUS),explicatio(CASUS);
    Nbar(NUM,GENUS,CASUS),[Nbar(gen)];
    Nbar(NUM,GENUS,CASUS),[comma],relsent(NUM,GENUS,CASUS).
```

The latter two are both exemplified in the sentence laudare te vult homo, aliqua portio creaturae tuae.

```
explicatio(CASUS):
    $PENALTY,[comma],NP(CASUS),[comma].
```

The numerals are either spelled as words from the lexicon or they are in the form of roman numerals in capital letters:

```
numerus(NUM,GENUS,CASUS):
  LEXQ(NUM,GENUS,CASUS);
  $SKIP("[MDCLXVI][MDCLXVI]*").
```

Finally, the NP may be composed of one or more Nbarbars. The formulation given here is defective, because NUM and GENUS should be influenced by the number and gender of elements.

```
NP(CASUS), NP(NUM,GENUS,CASUS):
   Nbarbar(NUM,GENUS,CASUS);
   Nbarbar(NUM,GENUS,CASUS),conj(co),NP(NUM,GENUS,CASUS).
```

The adjective phrase, again, is a generalized adjective. A participium or determinative pronoun may also serve as an adjective.

An adverb may precede as well as follow an adjective, but the former is preferred, to prevent spurious ambiguity of an adverb *between* two adjectives, and also because that feels intuitively right.

3.2 Verb Phrases

The verbal part of a sentence is very simple, since latin has no separate auxiliary verbs.

```
V(MODUS,PERS,NUM,VGENUS):
  LEXV(MODUS,PERS,NUM,TEMPUS,VGENUS);
  X,V(MODUS,PERS,NUM,VGENUS).
```

Some other verbal constructions:

```
tobe(MODUS,PERS,NUM,TEMPUS,VGENUS):
  [adject], TOBE(MODUS,PERS,NUM,TEMPUS,VGENUS).

infinitivus(VGENUS):
   LEXV(inf,TEMPUS,VGENUS), [object],[adject],
     (conj(co),LEXV(inf,TEMPUS,VGENUS);
      LEXV(inf,TEMPUS,VGENUS), clitic; ).


participium(NUM,GENUS,CASUS):
   [X],LEXV(part|pperf|gerund,NUM,GENUS,CASUS), [object].
```

Notice that no subcategorization information is available for the verbs, which causes needless overgeneration.

3.3 Sentence Structure

We adopt a very simple discourse structure: a sentence is composed of simple sentences glued together by certain separators. Three kinds of simple sentences are distinguished, statements, questions and commands, of which the latter are still poorly developed.

We distinguish between statements with an explicit main verb, and those with an (explicit or implicit) form of esse and a predicate.

```
statement:
   SVOC phrase;
   SxP phrase.
```

By an SVOC phrase we mean a phrase maximally containing Subject, Verb, Object and Complements in some order, the main verb being obligatory. This can be expressed in AGFL using the FWO operator as

```
[subject(PERS,NUM)] & V(MODUS,PERS,NUM,act) & [object] & [adject]
```

From this, topicalized versions (e.g. preposing an adjective from the subject or object) can be constructed.

The predicative sentences have many optional elements, including a predicate, but a form of esse must be present:

```
[subject(PERS,NUM,GENUS)]&TOBE(MODUS,PERS,NUM,TEMPUS,act)&
    [predicate(PERS,NUM,GENUS,nom)]&[adject]
```

The analysis of the discourse structure has to be refined on the basis of corpus study. The grammar is still incomplete, the wordlist contain errors and lacunae. It is hoped that some latin scholars will nevertheless find a use for the parser, and will extend and improve this work.

4 Preliminary Results

The parser generated from the grammar and lexicon described in the previous sections was applied to two corpora

- the Confessiones of St. Augustine (downloaded from [Augustinus Confessiones]), beautiful and well-polished latin
- Julius Caesar's accounts of the Gallic, Hispanic, African and Alexandrian wars, which consist of more rough-hewn political/military prose.

The lexicon was derived from the first corpus alone.

4.1 Coverage

In order to measure the coverage of the parser (number of words covered by the preferred analysis, divided by the total number of words in the text) we put at the root of the grammar a simple transduction, accepting either a sentence, which was then enclosed between square brackets, or an NP, which was enclosed between round brackets. Any word not covered by these was looked up in the lexicon. If it occurred in the lexicon with any category it was marked as SKIPped, and otherwise as UNKNown.

In the following table, we indicate the number of words in the text covered by the preferred analysis, the words from the lexicon not covered by the analysis, and the words not occurring in the lexicon.

corpus	number of words	covered	skipped	unknown
Augustinus	78784	65264 (82.8%)	8763 (11.1%)	4757 (6.0%)
Caesar	49961	37634 (75.3%)	7990 (16.0%)	4337 (8.7%)

4.2 Speed

We also measured the CPU time needed to obtain the preferred analysis for each segment on a 700Mhz INTEL PC on the two corpora.

corpus	total time	words parsed / second
Augustinus	3 min 51 sec	341
Caesar	2 min 1 sec	413

Since processors four times as fast are easy to find, this may also well be the fastest Latin parser ever constructed! Most of the time is actually spent in lexicalization (lexicon lookup, robust recognition of proper names).

5 Conclusions

The Latin grammar and lexicon were developed by one person in a few lost weeks between Christmas and the Spring term. This was of course only possible by the availability of good tools - the AGFL formalism, a specialized parser generator for Natural Language parsing, UNIX tools for text transformations - and a good schooling in classical latin. Of course quite a lot of improvement and error correction is still needed.

However, the generative technique employed to produce the lexicon is suitable for any language with a complicated morphosyntactic structure. The trie-based lexicon system of AGFL has no problems with millions of wordforms, so that even highly inflected languages lend themselves to generative lexicon production and maintenance. The AGFL formalism, which was developed for describing English turned out to be very suitable for the compact description of Latin, including its free-word-order aspects.

The parser, latin grammar and lexicon described here are freely available from the [AGFL website].

References

[Koster, 1991] Cornelis H.A. Koster, Affix Grammars for Natural Languages. In: H. Alblas and B. Melichar (Eds.), *Attribute Grammars, applications and systems.* SLNCS 545, Heidelberg, 1991, pag. 469-484.

[linguistics textbook] `http://www.u-grenoble3.fr/lebarbe/Linguistic_Lexicon/`

[Redde Rationem] A.G. de Man et al (1979), *redde rationem - recapulationes*, Wolters - Noordhoff.

[Latijnse Leergang] S.F.G. Rotteveel Mansfeld en R. Waleson (1970), /em Latijnse leergang, tweede druk, Wolters - Noordhoff.

[Augustinus Confessiones] `http://www.thelatinlibrary.com/august.html`

[another latin parser] `http://www.levity.com/alchemy/latin/latintrans.html`

[AGFL website] `http:www.cs.kun.nl/agfl`