Practical 2: Grammar engineering

This practical is worth 50% of the coursework credit for this module. Its due date is Tuesday 5th of April 2022, at 21:00. The usual penalties for lateness apply, namely Scheme B, 1 mark per 8 hour period or part thereof.

The purpose of this assignment is to make you familiar with specifications of syntax. You will be engineering a grammar for a small subset of English. For your convenience, this task is broken up into steps. At first, the grammar will be a pure context-free grammar and later this will be refined to become a feature (or 'unification') grammar.

As in the first practical, we will be using NLTK with Python3.

Step 0: getting started

Investigate the provided files:

- `parse.py`
- `grammar.fcfg`
- `positives.txt`
- `negatives.txt`

The first is a small Python program that compiles a parser from a feature grammar, and applies it on two files, one with positive examples and one with negative examples. Have a close look at how ARG is used for arguments of verbs. Keep this treatment of arguments in what follows, so we may later implement subcategorisation in an elegant way. Here we assume that "`loves`" requires exactly one argument; hence the two negative examples, with zero and two arguments, respectively.

Run `/usr/local/python/bin/python3 parse.py` (or just `python3 parse.py` or `python parse.py` if working on your own machine) and see what happens.

Step 1: lexicon and context-free grammar

Consider the following positive examples:

```
Guinevere sighs
Guinevere and Arthur sigh
horses drink water
Arthur rides the horse
Guinevere rides on Mondays
Arthur rides
Guinevere rides the old white horse near the castle on Mondays
when Arthur drinks beer Guinevere sighs
Guinevere thinks Arthur drinks water
Arthur rides near the castle and drinks from the chalice
Arthur rides and drinks beer and water
when do Arthur and Guinevere drink beer
when does Guinevere ride
```

Note that all punctuation has been removed in order to avoid complications, and we do not enforce capitalisation at the beginning of sentences.

Extend `grammar.fcfg` with more rules so that all words from the above sentences are included. You will need to introduce more parts of speech such as `Det` (e.g. "`the`"), `Prep` (e.g. "`near`", "`from`"), `Adj` (e.g. "`white`"), and a few more. Note that the two occurrences of "`when`" have different functions, so need to be associated with different parts of speech.

At this point, you may not want to distinguish between singular and plural noun phrases, nor between different verb forms, nor between verbs with different subcategorisation frames.

Also add more context-free rules, so that the above sentences can be derived. Make sure the rules defining the start symbol (`S`) come first. (NLTK by default assumes that the first mentioned nonterminal is the start symbol.)

When designing your grammar, beware of the distinction between argument and adjunct. In the above example sentences, PP "`from the chalice`" is arguably an argument while PP "`on Mondays`" is an adjuct.

Step 2: intermediate testing of the grammar

Add the above positive examples to `positives.txt` and add the below negative examples to `negatives.txt`.

```
when Arthur drinks
when does Guinevere
```

```
the horses drinks water
Arthur thinks the horse
```

(Some of the above might be grammatical in special contexts, e.g. assuming 'ellipsis', i.e. omitted phrases that are understood from the larger context of a dialogue. In this practical, we don't consider ellipsis, nor do we consider uncommon usage of words that would require a stretch of the imagination to justify.) Once more run `/usr/local/python/bin/python3 parse.py`.

Step 3: feature grammar

The grammar you wrote in Step 1 will likely accept some of the negative examples. This is because the following have not been modelled:

- number agreement,
- subcategorisation.

For number agreement, remember that English has five verb forms for ordinary verbs (and a few more for **to be**):

- *base form*: **to write**, **you write**
- *third person singular present*: **he writes**
- *preterite* (a.k.a. simple past): **wrote**
- *past participle*: **written**
- *present participle* (a.k.a. gerund if used as noun): **writing**

For our simple examples, we only need the first form (used for third person plural present, and infinitive) and the second (third person singular present). Features can be added to the grammar to ensure that only the correct verb forms are allowed, and that there is number agreement for those verb forms where it is relevant.

Subcategorisation should be implemented as illustrated by the following example (which ignores the issue of number agreement):

```
S -> NP VP[SUBCAT=nil]
VP[SUBCAT=?rest] -> VP[SUBCAT=[HEAD=?arg, TAIL=?rest]] ARG[CAT=?arg]
VP[SUBCAT=?args] -> V[SUBCAT=?args]

ARG[CAT=np] -> NP
ARG[CAT=pp] -> PP
```

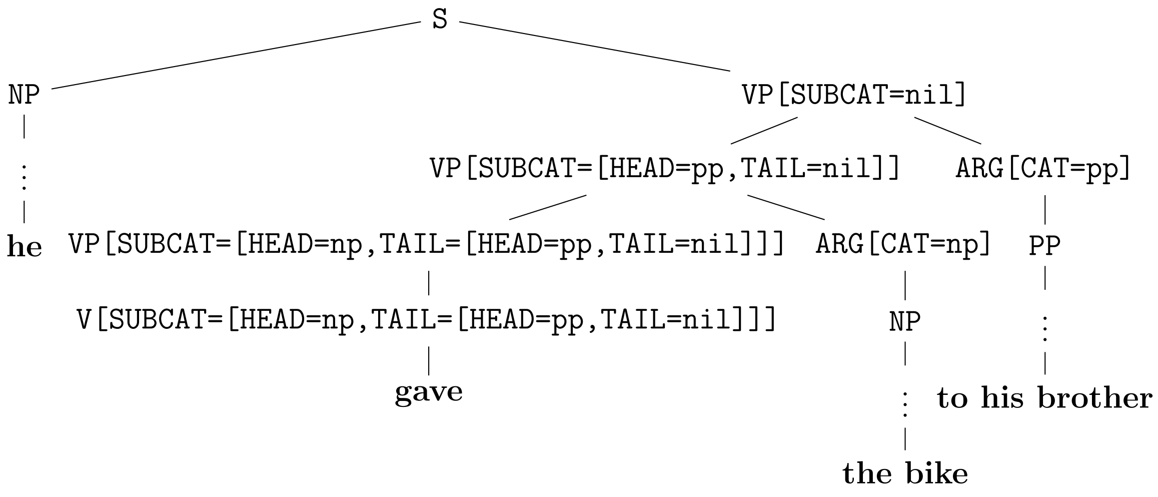
S
NP
VP[SUBCAT=nil]
VP[SUBCAT=[HEAD=pp,TAIL=nil]]
ARG[CAT=pp]
he
VP[SUBCAT=[HEAD=np,TAIL=[HEAD=pp,TAIL=nil]]]
ARG[CAT=np]
PP
V[SUBCAT=[HEAD=np,TAIL=[HEAD=pp,TAIL=nil]]]
NP
gave
to his brother
the bike

Figure 1: Graphical representation of the parse of **he gave the bike to his brother**. Note the two applications of `VP[SUBCAT=?rest] -> VP[SUBCAT=[HEAD=?arg, TAIL=?rest]] ARG[CAT=?arg]`. Also note the topmost `VP` has `[SUBCAT=nil]`, which is needed to apply the rule with left-hand side `S`.

```
V[SUBCAT=nil] -> 'sneezed'
V[SUBCAT=[HEAD=np, TAIL=[HEAD=pp, TAIL=nil]]] -> 'gave'
```

Figure 1 illustrates how the rules for subcategorisation are applied, for a subcategorisation frame of "`gave`" with two arguments, namely a `NP` and a `PP`. In order to handle the verbs in our example sentences, further rules for `V` and `ARG` are needed, but it should be possible to reuse the rule `VP[SUBCAT=?rest] -> VP[SUBCAT=[HEAD=?arg, TAIL=?rest]] ARG[CAT=?arg]` for several verbs, regardless of their subcategorisation frames.

Step 4: final testing

Again test the positive and negative examples, and verify that all positive examples are accepted, and none of the negative examples are accepted. You may add more positive and negative examples (with words in the lexicon) to convince yourself that your grammar is satisfactory.

Requirements

Submit a zipped file containing:

- `parse.py` (unmodified)
- `grammar.fcfg` (extended by you)
- `positives.txt` (extended by you)
- `negatives.txt` (extended by you)
- a report in PDF

The report should contain:

- Description of, and motivation for, any interesting choices you have made in engineering the grammar, for example the choice of the set of categories.
- Critical reflection on the language the grammar accepts. Does it accept any sentences that you would consider ungrammatical with regard to standard English?
- Any other thoughts on this practical.
- Explicit mention of contributions that you consider to be extensions (see below).

Marking and extensions

Marking is according to the school handbook. The basic requirements above earn you up to 17 marks if all is done well; this will need to include implementation of number agreement, verb forms and subcategorisation as outlined above. If only a context-free grammar is produced, without features that implement number agreement, verb forms or subcategorisation, then no more than 12 marks can be attained. Marks higher than 17 require extensions that contribute to demonstrating your understanding of grammars for natural language. Possible extensions include the implementation of new types of sentences, and their discussion in the report, such as:

```
horses love drinking water
Arthur may have loved Guinevere
Arthur gives Guinevere the horse
what does Arthur give Guinevere
whom does Guinevere give the horse
whom do Guinevere and Arthur give the horse
what does Guinevere drink
whom does Guinevere think Arthur gives the horse
what does Guinevere think Arthur drinks
```

Finding a general and elegant solution to handle sentences of the last six types above is quite challenging, as it requires processing the subcategorisation frames in a novel manner; ideally, you would not need to add new (lexical) rules for verbs.

Hints

- Try to avoid that your feature grammar allows sentences that are syntactically incorrect. Do not worry however about accepting sentences with nonsensical meaning; e.g. we consider "`the water sighs`" to be perfectly acceptable from a syntactic viewpoint.
- We generally prefer small numbers of simple rules generating many different (correct) sentences over large numbers of rules that each capture few cases. If your grammar can handle only the given positive examples and little more, then this is not very satisfactory. For example, the largest number of adjectives we see in any example sentence in Step 1 is two, but why should we not allow sentences with three or more adjectives in front of a noun?
- Ambiguity is unavoidable in natural language. It is fine if your grammar allows several parses for a single sentence, provided these different parses correspond to different interpretations.
- There may be more than one solution to achieve roughly the same language, but a grammar that uses commonly accepted category names (see e.g. lecture notes) is preferable over one that does not.

Pointers

- Marking
  `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`
- Lateness
  `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`
- Good Academic Practice
  `https://www.st-andrews.ac.uk/students/rules/academicpractice/`