Assignment 2: Grammar Engineering

NLP-BIU | Autumn 2022

Due: 4 January 2023

Credit due: This assignment is based on an assignment by Jason Eisner at JHU.

Goal

This assignment will help you understand how Context-Free Grammars (CFGs) work, and how they can be used — sometimes comfortably, sometimes not so much — to describe natural language. It will also make you think about some linguistic phenomena that are interesting in their own right.

Deliverables

Your final submission should include a single zip file containing a file `ANSWERS.pdf` (specified below) as well as the files `generate.py, grammar1, grammar1.gen, grammar2, grammar2.gen, part3.gen, grammar4, grammar5`, and optionally, `generatex.py`, as specified in the assignment. Make sure your `ANSWERS.pdf` file includes your names and IDs at the top line.

You will submit a file in PDF format called `ANSWERS.pdf`. This file should contain your name and ID number. Whenever the assignment says "discuss.." or "describe.." it means that you should write a discussion or a description in that file. Make sure your `ANSWERS.pdf` file is clearly organized, and indicates clearly which question you are answering.

Part 0 | Generating Sentences from a CFG (1pt)

In this assignment, we provide you with the following components:

- A basic grammar file grammar. Make sure to review its format below.
- a python program generate.py that generates sentences from a grammar.

The program `generate.py` takes a grammar file as an argument, and generates a sentence from it. Your task is to generate sentences from the grammar using:

```
python generate.py grammar
```

The Grammar File: The format of the grammar file is as follows:

```
# A fragment of the grammar to illustrate the format.
 1 ROOT S .
 1 S NP VP
 1 NP Det Noun # There are multiple rules for NP.
 1 NP NP PP
 1 Noun president
 1 Noun chief of staff
```

These lines correspond to the following rules (in this order):

ROOT → S

S → NP VP

NP → Det Noun

NP → NP PP

Noun → president

Noun → chief of staff

Notice that a line consists of three parts:

- a number (ignore this for now)
- a nonterminal symbol, called the "left-hand side" (LHS) of the rule
- a sequence of zero or more terminal and nonterminal symbols, which is called the "right-hand side" (RHS) of the rule

A hash sign (#) indicates a comment, and everything after it is ignored.

We take ROOT to be the root of the grammar.
That is, every generation must start with a ROOT.

The Program File: Look at the `generate.py` program and **understand** it.

Extend the program so that it accepts an optional argument (indicated by `-n`) that specifies the number of sentences to generate. For example, the invocation: `python generate.py grammar -n 5` will generate 5 sentences, the invocation `python generate.py grammar -n 2` will generate two sentences, and `python generate.py grammar` will generate one sentence, as it does now. Each sentence should be printed on its own line.

Note, the list sys.argv includes the command line arguments of a python program. You can either take care of the command line processing logic yourself, or use a module such as docopt.)

Part 1 | Weights (9pts)

Consider the output sentences you generated in part 0 and answer the following questions:

1. Why does the program generate so many long sentences? Specifically, what grammar rule is responsible for that and why? What is special about this rule? discuss.
2. The grammar allows multiple adjectives, as in: "the fine perplexed pickle". Why do the generated sentences show this so rarely? discuss.
3. The grammar format allows specifying different weights for different rules.

   For example, in the grammar:

   ```
   3 NP A B
   1 NP C D E
   1 NP F
   2 X NP NP
   ```

   The NP rules have a relative ratio of 3:1:1, so, when generating an NP, the generator will pick the first rules 3/5 of the times, the second one 1/5 of the times, and the third one 1/5 of the times.

   Which numbers must you modify to fix the problems in (1) and (2), making the sentences shorter and the adjectives more frequent?

   Verify your answer by generating from the grammar. Discuss your solution (which rules did you modify, and why).
4. What other numeric adjustments can you make to the grammar in order to favor a set of more natural sentences? Experiment and discuss.

What to Submit:

- Hand in your grammar file (including the solution to (3) above), with comments that motivate your changes, in a file named `grammar1`
- Hand in 10 sentences generated via `grammar1`, in a file called `grammar1.gen`.

Part 2 | Extending the Grammar (30pts)

Modify the grammar so that it can also generate the types of phenomena illustrated in the following sentences:

(a) Sally ate a sandwich .

(b) Sally and the president wanted and ate a sandwich .

(c) the president sighed .

(d) the president thought that a sandwich sighed .

(e) it perplexed the president that a sandwich ate Sally .

(f) the very very very perplexed president ate a sandwich .

(g) the president worked on every proposal on the desk .

(h) Sally is lazy .

(i) Sally is eating a sandwich .

(j) the president thought that sally is a sandwich .

You want to end up with a single grammar that can generate *all* of the sentences in part 1 as well as *all* sentences (a)–(j) here. Furthermore, we expect your grammar to generate sentences which do not exist in this list but have grammatically similar structures.

While your new grammar may generate some very silly sentences, it should not generate any that are strictly ungrammatical. For example, in the following pair, your grammar must be able to generate the first sentence (g), but it must exclude the second one (h) marked with an asterix * .[1]

(g) the president thought that a sandwich sighed .

(h) * the president thought that a sandwich sighed a pickle.

More specifically, your grammar should respect the *subcategorization frame* of verbs (number and types of required arguments, as defined in class). Think, for instance, about VP taking an S and not NP. However it doesn't have to respect *selectional restrictions* (the semantic restrictions on the kind of arguments). That is, no need to distinguish between nouns that can eat, want, or think and ones that cannot.

Overall, while the sentences should be okay structurally, they don't need to really make sense.

Notes:

- Be sure you can handle the particular examples we suggested here, which means, among other things, that your grammar must include the words in those examples.
- Make sure that your grammar can generalize from the suggested example sentences. For example, sentence (b) invites you to think about the ways in which conjunctions ("and", "or", etc) can be used in English. Sentence (g) invites you to think about how prepositional phrases ("on the desk", "over the rainbow", "of the United States") look like, and how they are used in English sentences.

[1] In linguistics, * is traditionally used to mark ungrammatical sentences.

- Furthermore, note that handling sentences (b) and (h)/(i) can interact in a bad way, to create ungrammatical sentences. You do not need to solve this issue in this part of the assignment, but you do need to discuss it and explain what the problem is, using an example and a short explanation.
- Finally, an important technical note. The grammar file allows comments and whitespace because the grammar is really a kind of specialized programming language for describing sentences. Throughout this assignment, **when writing grammars, you should strive for the same level of elegance, generality and documentation as when writing code.**

Implementation Hints : If you want to see the effect of various rules while developing your grammar, you can give them a very high weight so that they will trigger often. Implementing Part 3 can also help you debug your grammar.

What to submit: Briefly discuss your modifications to the grammar. Hand in the new grammar (commented) as a file named `grammar2` and about 20 random sentences that illustrate your modifications in a file named `grammar2.gen`.

How your grammars will be tested: We want to see that your grammar can produce sentences with structures as we specified, and generalize to similar structures, but not produce bad (i.e., ungrammatical) sentences. We will run a parser that uses your grammar to try and parse a set of sentences you have not previously seen. Your parser should be able to parse the grammatical ones, and not be able to parse the ugrammatical ones. **Recall,** you can parse a sentence with a grammar if and only if this sentence can be generated by the grammar.

Part 3 | Tree Structures (10pts)

Modify the generate.py program so that it gets an optional command line switch -t. When -t is present, the program should generate not only sentences, but also the tree structures that generates the sentences. For example, when invoked as:

```
python generate.py grammar -t
```

instead of just printing `the floor kissed the delicious chief of staff`. it should print the more elaborate version

```
(ROOT (S (NP (Det the)
             (Noun floor))
         (VP (Verb kissed)
             (NP (Det the)
                 (Noun (Adj delicious)
                       (Noun chief
                             of
                             staff)))))
      .)
```

which includes extra information showing how the sentence was generated. For example, the above derivation used the rules Noun → floor and Noun → Adj Noun, among others.

Note: It's not too hard to print the pretty indented format above. But it's not necessary. It's sufficient if you will the entire expression on a single line:

```
(ROOT (S (NP (Det the) (Noun floor)) (VP (Verb kissed) (NP (Det the)
(Noun (Adj delicious) (Noun chief of staff))))) .)
```

The bracketed expression can be visualized in tree form using web-based visualizers like this and this and this, or java-based visualizers like this. This sort of output can be useful when debugging your grammar – understanding which rules are responsible for what structures.

Implementation Hint: The changes to the original code are not big. You don't have to represent a tree in memory, so long as the string you print has the parentheses and non-terminals in the right places.

Your code should support both the -n from part 0, and the -t switch, either alone or together.

What To Submit: Generate 5 more random sentences, in tree format. Submit them as in a file called `part3.gen` as well as the code for the modified `generate.py` program.

Part 4 | Additional Linguistic Structures (40pts, 20 each)

Think about all of the additional linguistic phenomena presented below, and extend your grammar from part 2 to handle TWO of them – any two your choice. Briefly discuss your solutions and provide sample output.

Be sure you can handle the particular examples suggested herein, which means among other things that your grammar must include the words in those examples. You should also generalize appropriately beyond these examples.

As always, try to be elegant in your grammar design, but you will find that these phenomena are somewhat harder to handle elegantly with CFG notation.

(a) **"a" vs. "an"**.
Add some vocabulary words that start with vowels, and fix your grammar so that it uses "a" or "an" as appropriate (e.g., "an apple" vs. "a president"). This is harder than you might think: how about "a very ambivalent apple"?

(b) **Yes-no questions**. Examples:

did Sally eat a sandwich ?

will Sally eat a sandwich ?

Of course, don't limit yourself to these simple sentences. Also consider how to make yes-no questions out of the statements in part 2.

(c) **Relative clauses.** Examples:

the pickle kissed the president that ate the sandwich .

the pickle kissed the sandwich that the president ate .

the pickle kissed the sandwich that the president thought that Sally ate .

Of course, your grammar should also be able to handle relative-clause versions of more complicated sentences, like those in part 2.

Hint: These sentences have something in common with (d).

(d) **WH-word questions**.

If you also did (b), handle questions like

what did the president think ?

what did the president think that Sally ate ?

what did Sally eat the sandwich with ?

who ate the sandwich ?

where did Sally eat the sandwich ?

If you didn't also do (b), you are allowed to make your life easier by instead handling "I wonder" sentences with so-called "embedded questions":

I wonder what the president thought .

I wonder what the president thought that Sally ate .

I wonder what Sally ate the sandwich with .

I wonder who ate the sandwich .

I wonder where Sally ate the sandwich .

Of course, your grammar should be able to generate wh-word questions or embedded questions that correspond to other sentences.

Hint: All these sentences have something in common with (c).

(e) **Singular vs. plural agreement.**

For this, you will need to use a present-tense verb, since past tense verbs in English do not show agreement. Examples:

the citizens choose the president .

the president chooses the chief of staff .

the president and the chief of staff choose the sandwich .

(You are not allowed to select both this question (e) and question (a), as the solutions are somewhat similar.)

(f) **Tense and Aspect.**

Expression tense information may involve more than a single verb. For example, "the president has been eating a sandwich" . Here, you should try to find a reasonably elegant way of generating all the following 12 tense (present, past, future) × aspect (simple, perfect, progressive, perfect-progressive) combinations:

aspect↓ tense→	present	past	future
Simple	eats	ate	will eat
Perfect	has eaten	had eaten	will have eaten
Progressive	is eating	was eating	will be eating
Perfect Progressive	has been eating	had been eating	will have been eating

(g) **Appositives.** Examples:

the president perplexed Sally , the fine chief of staff .

Sally , the chief of staff , 59 years old , who ate a sandwich , kissed the floor .

The tricky part of this one is to get the punctuation marks right. For the appositives themselves, you may rely on some canned rules such as "Appos → 59 years old". However, if you also did (c), try to extend your rules from that problem to automatically generate a range of appositives such as who ate a sandwich and which the president ate.

What To Submit: Hand in your grammar (commented) as a file named `grammar4`. The first line of `grammar4` must be a comment, including the letters of the two phenomena you chose to implement, separated by space. Example:

```
# b f
```

In addition, provide a description of your solution in `ANSWERS.pdf`.

Important Note: Your final grammar should handle everything from part 2, plus both of the phenomena you chose to add. This means you have to worry about how your rules might interact with one another. Good interactions will elegantly use the same rule to help describe two phenomena. Bad interactions will allow your program to generate ungrammatical sentences, which will hurt your grade.

Part 5 | Extra (up to 15pt)

Impress us! How much more of English can you describe in your grammar?

Extend the grammar in some interesting way and tell us about it. For ideas, you might look at some random sentences from a magazine. Name the grammar file `grammar5`.

If it helps, you are also free to extend the notation used in the grammar file as you see fit, and change your generator accordingly. If so, name the extended generator `generatex.py`. Provide a discussion of your changes in `ANSWERS.pdf`.

Good Luck!