DD2417 Language Engineering
Assignment 1

Johan Boye
2024-03-18

All programs should be written in Python 3, unless specified otherwise in the problem instructions. Don't use any external libraries (that are not part of the Python 3 distribution) unless otherwise specified. Problems 1–4 are pen-and-paper problems.

Mandatory part

1. Find all the syntax trees derivable from the sentence *"He built the box with a hammer in the yard behind the house"*, and the grammar:

S	→	NP VP	Pron	→	he
NP	→	Pron	Verb	→	built
NP	→	Det Noun	Prep	→	with
NP	→	Det Noun PP	Prep	→	in
VP	→	Verb NP	Prep	→	behind
VP	→	Verb NP PP	Noun	→	hammer
PP	→	Prep NP	Noun	→	box
Det	→	the	Noun	→	yard
Det	→	a	Noun	→	house

   For each tree, explain what a semantically sensible interpretation of the tree might be (if there is a sensible interpretation). In particular: Where is the box? Where is the hammer? Where is the yard?

2. Consider the grammar:

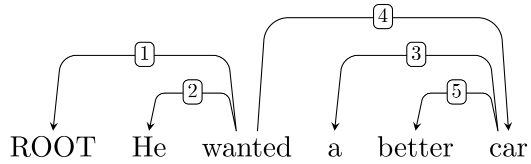
S	→	NP VP	Noun	→	faces
NP	→	Det Adj Noun	Noun	→	suspect
NP	→	Adj Noun	Noun	→	pressure
NP	→	Det Noun	Adj	→	increased
NP	→	Noun	Adj	→	suspect
VP	→	V NP	V	→	faces
Det	→	the	V	→	increased

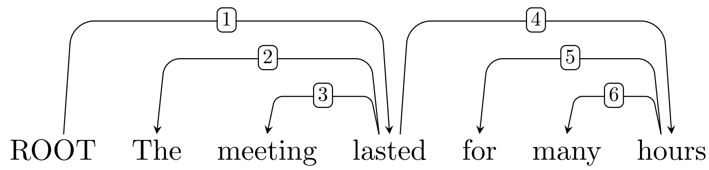
   (a) Convert the grammar into a weakly equivalent Chomsky normal form grammar.

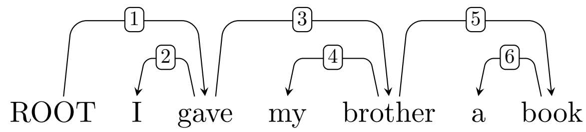
   (b) Using your new grammar, use the CKY algorithm to parse the sentence "*the suspect faces increased pressure*". Show your completed parse table as result.

   (c) How many syntax trees for the entire sentence can you find? Draw these syntax trees, and explain how you can retrieve them from the parse table.

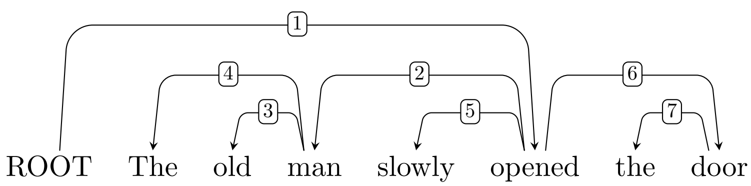
3. Each of these dependency trees has one edge which is incorrect. Decide which one, and explain how it should be drawn instead.

   (a) 

   (b) 

   (c) 

4. Below is a correct dependency tree, but with the labels missing. For each of the labels 1–7, determine the appropriate relation label. (`https://universaldependencies.org/u/dep/` has a list of all labels).



5. (Dependency parsing) Transition-based parsing is an efficient way of producing dependency trees from a sentence. Your task in this problem is to fill in some missing parts of a transition-based dependency parser.

   (a) First go to the `DepParser` folder, and type:

   ```
   pip install -r requirements.txt
   ```

   Now complete the method `valid_moves` in the `Parser` class so that it, given a parser configuration (the contents of the buffer, the stack, and the partially built tree), returns the list of valid moves (shift (SH), left-arc (LA), right-arc (RA)) in that configuration.

   (b) Complete the method `move` so that it, given a parser configuration, returns the resulting configuration after the move has been carried out (the new contents of the buffer and the stack, and the new partially built tree). After you have done this, run `step_by_step.sh` (or `.bat`) to make sure that it works.

(c) Finally, extend the method `compute_correct_move` so that it, given a parser configuration and the correct final tree, computes the correct move for the parser to make in that configuration. Run the script `compute_correct_move.sh` (or `.bat`), and compare the output to the file `correct_moves_en-ud-dev.txt`.

Optional part

6. (CKY parsing) The CKY algorithm is an efficient method of analyzing sentences according to a grammar in Chomsky Normal Form (CNF).

(a) First go to the CKY folder, and type:

```
pip install -r requirements.txt
```

Now extend the method `parse` in the `CKY` class so it produces a CKY parse table from an input sentence. For instance, running the script `run_cky_parser_1`, which parses the sentence "*giant cuts in welfare*" given the grammar in the file `grammar.txt`, should result in:

```
+--------------+----------------+----------+--------------------+
| ['NP', 'JJ'] | ['NP']         | []       | ['S', 'NP', 'NP'] |
| []           | ['NP', 'Verb'] | []       | ['NP', 'VP']       |
| []           | []             | ['Prep'] | ['PP']             |
| []           | []             | []       | ['NP']             |
+--------------+----------------+----------+--------------------+
```

(b) Extend the method `print_trees` so that it prints all parse trees derivable from a certain cell in the parse table, rooted with a given symbol. For instance, the two trees derivable from the topmost rightmost cell, rooted with 'NP', are

```
NP(JJ(giant), NP(NP(cuts), PP(Prep(in), NP(welfare))))
NP(NP(JJ(giant), NP(cuts)), PP(Prep(in), NP(welfare)))
```

and the only tree derivable from the same cell, rooted with 'S', is:

```
S(NP(giant), VP(Verb(cuts), PP(Prep(in), NP(welfare))))
```

(c) Create a new grammar file `grammar2.txt` containing the grammar you constructed in problem 2(a). Use your program to create a parse table for the sentence "*the suspect faces increased pressure*", and to print out all the parse trees for that sentence.