Exercises on Syntactic Parsing

Ion Androutsopoulos, 2022–23

There are no exercises to be handed in as assignments in this part of the course.

1. (a) Draw the two (phrase structure) parse trees that would be obtained for the English sentence "we saw the doctor with the white coat", when the following Context Free Grammar (CFG) is used, with start symbol S.

V → saw
Det → the
N → doctor | coat
Prep → with
Adj → white
Pron → we

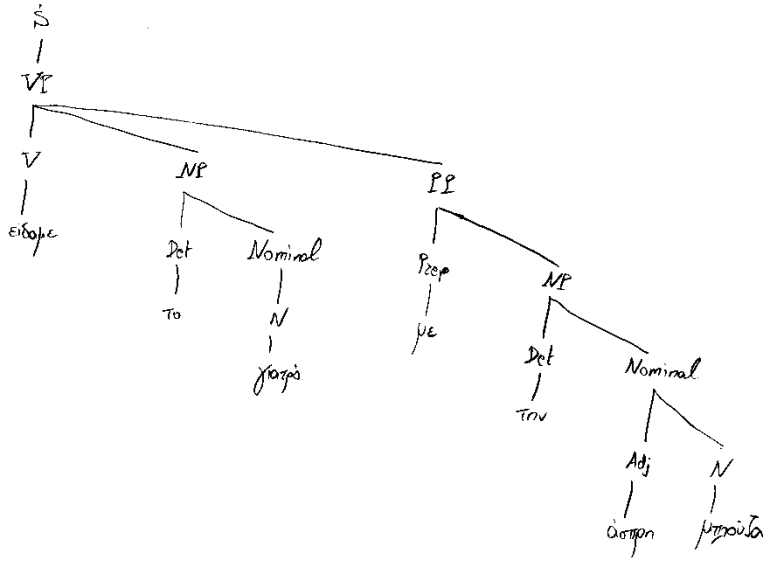
Nominal → N | Adj N | N PP
NP → Det Nominal | Pron
PP → Prep NP
VP → V NP | V NP PP
S → NP VP

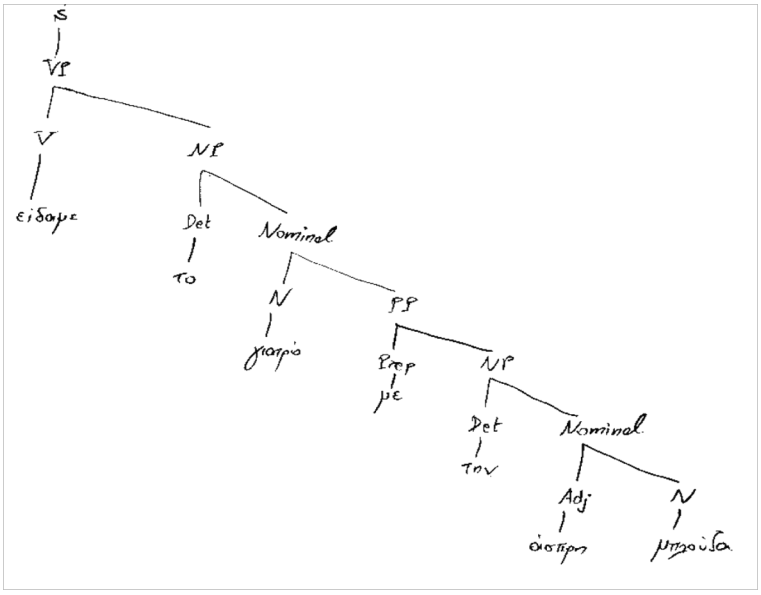
(b) Repeat for the corresponding Greek sentence "είδαμε το γιατρό με την άσπρη μπλούζα" ("[we] saw the doctor with the white coat"), when the following CFG is used.

V → είδαμε
Det → το | την
N → γιατρό | μπλούζα
Prep → με
Adj → άσπρη

Nominal → N | Adj N | N PP
NP → Det Nominal
PP → Prep NP
VP → V NP | V NP PP
S → VP

Answer: The trees for the Greek sentence are shown below. The English trees are very similar, but the S has the form NP VP with the NP being the Pron (pronoun) "we".





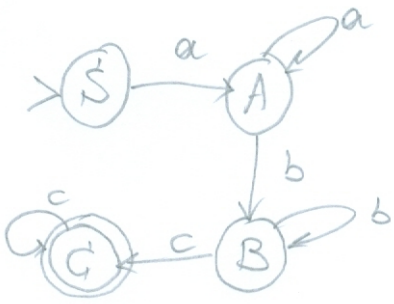
2. (Optional) (a) Write a right-linear regular grammar for the language $a^m b^n c^k$, $m > 0$, $n > 0$, $k > 0$.

Answer: The grammar is shown below. The start symbol is S.

S → aA
A → aA
A → bB
B → bB
B → cC
C → cC
C → ε

(b) Draw a Finite State Automaton (FSA) to recognize (or generate) the same language.

Answer: The FSA is shown below. A ">" marks the initial state, and the final state is shown with a double circle.



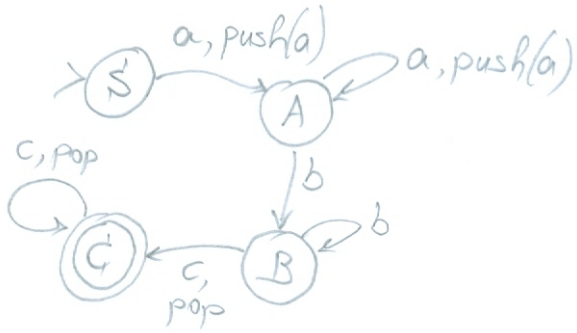
(c) Write a CFG for the language $a^m b^n c^m$, with $m > 0$, $n > 0$.

Answer: The grammar is shown below. The start symbol is S.

S → aBc
S → aSc
B → bB
B → b

(d) Draw a Push-Down Automaton (PDA) to recognize (or generate) the language of the previous sub-question.

Answer: The FSA is shown below. A ">" marks the initial state, and the final state is shown with a double circle.



3. (a) Convert the following CFG to Chomsky Normal Form (CNF):

S → NP VP
NP → Det N
VP → Vintrans | Vtrans NP
Det → the
N → customer | flight
Vtrans → cancelled | confirmed
Vintrans → departed | arrived

Answer:

S → NP VP | NP Vintrans
NP → Det N
VP → Vtrans NP
Det → the
N → customer | flight
Vtrans → cancelled | confirmed
Vintrans → departed | arrived

(b) Show how CKY would parse sentences like "the customer confirmed the flight" or "the flight departed" using the grammar above. Draw and fill in CKY's table, as in the slides.

(c) Explain exactly how CKY could be modified (slide 19) to produce also parse trees. Show how the parse tree would be produced using examples for the grammar above.

(d) Confirm that the time complexity of the CKY parsing algorithm is $O(n^3)$, where n is the length (number of tokens) of the input sequence.

Answer to part (d): We have to fill in the $O(n^2)$ cells of the table. For each cell, we need to check the $O(n)$ possible splits (in two parts A, B to be combined by a rule $X \rightarrow A\ B$) of the part of the input sentence that corresponds to the particular cell. For each candidate split (in two parts A, B), we need to scan the rules of the grammar (excluding lexicon rules) to find the corresponding rules (of the form $X \rightarrow A\ B$). If the grammar has $|G|$ rules (excluding lexicon rules) and checking the applicability of each rule costs $O(1)$, then each scan of the grammar

costs $O(|G|)$ and the overall complexity becomes $O(n^3|G|)$.[1] If we ignore the size of the grammar, then the overall complexity is $O(n^3)$.

4. (Optional) (a) Implement CKY, assuming that the input grammars will be CFGs already in CNF. Your implementation must read the input grammar from a plain text or XML file and print the decision of the CKY parser (accept, reject). You may represent the input grammars in any format you prefer, but it must be easy to modify the grammars with a plain text editor. (b) Extend your implementation to print the parse trees. You may print the trees in any format you prefer, but the format must be reasonably easy to understand. (c) Demonstrate the functionality of your implementation using your own grammars or grammars from the slides.

5. Use the dependency parser of spaCy or Stanford's Stanza to obtain the dependency trees of a few English and/or Greek sentences.[2] You may also want to check out gr-nlp-toolkit.[3]

8. (Optional) Implement a simplistic version of a transition-based dependency parser (slides 22–27) and/or a graph-based dependency parser (slides 30–32) using TensorFlow/Keras or PyTorch. You may use a greedy decoder in the graph-based parser. You only need to produce unlabeled dependency trees. Assume that the correct (gold) POS tags of the input sentence are given to the parser. Train and test your parser on a treebank from the Universal Dependencies Project for a particular language (e.g., English or Greek) of your choice.[4] Use pre-trained word embeddings[5] or a BERT-like model to obtain context-aware embeddings.[6] Evaluate your parser by computing its UAS (slide 28).

[1] Consult also https://en.wikipedia.org/wiki/CYK_algorithm. This answer is based on more detailed comments by Michail Vazaios (thanks!).
[2] See https://spacy.io/ and https://stanfordnlp.github.io/stanza/.
[3] See http://nlp.cs.aueb.gr/software.html.
[4] See http://universaldependencies.org/.
[5] E.g., from https://nlp.stanford.edu/projects/glove/ or https://fasttext.cc/.
[6] Consult the BSc theses of C. Dikonimaki and N. Smyrnioudis at http://nlp.cs.aueb.gr/theses.html.