Homework Lab 2 - syntactic parsing

Assignment (3p)

Deadline: 3 weeks (31.03.2022)

Implement a parser in python, without using the NLTK (or another library)'s functionality for parsing (you can use NLTK for tokenizing for example).
You can choose to implement **one of** the parsers described in class.

The program will receive as input a "grammar.txt" file and a sentence to parse, and output a parse tree (encoded in the same fashion as the parse trees produced by NLTK as seen during the lab).
You can use the sample grammar provided below (alternatively, you can also test it on one of the grammars used during the lab), and test it on the test sentences to produce a parse tree:

```
S -> NP VP
S -> VP
NP -> DT NN
NP -> DT JJ NN
NP -> PRP
VP -> VBP NP
VP -> VBP VP
VP -> VBG NP
VP -> TO VP
VP -> VB
VP -> VB NP
NN -> "show" | "book"
PRP -> "I"
VBP -> "am"
VBG -> "watching"
VB -> "show"
DT -> "a" | "the"
MD -> "will"
```

Test sentence: `I am watching a show.`
Desired output:

```
(S
  (NP (PRP I))
  (VP (VBP am) (VP (VBG watching) (NP (DT a) (NN show)))))
```

Share your results via Google Colab with ana-sabina.uban@unibuc.ro (if you prefer, send the *.py files by email to ana-sabina.uban@unibuc.ro)

More details on some possible parser implementations below:

Details

Left corner parser

The Left-Corner Parser uses both a top-down and a bottom-up strategy. In the top-down steps it tries to predict the possible structure of the phrase, however continuously checking the phrase structure through a bottom-up process (checking if the input phrase matches the prediction). A left-corner parser does some preprocessing before the parsing itself. It creates an association between each non-terminal label and a list of all possible left corners (start of the expression). Before applying a production from the context free grammar, it searches for the next word that there is one starting label (in the left corners list) that applies to it.

Algorithm steps:

1. Create the left corner table. For each production, A -> sequence_of_symbols we take the nonterminal A and check if it already has a corresponding line in the table. If it doesn't, we create a new line for it, and add the first (leftmost) symbol from the sequence_of_symbols. If we already have a row associated to A, we just add the symbol, to that row.
2. After we've created the table, we start to compute the parsing. We consider the starting variable to be the symbol S (sentence). In order to find a correct production for the symbol S, we do a bottom-up search, using the left corner table. We start from the first input word and move it from the input string into a list of already processed words. We find for it, in the left corner table, the first line that contains it. We take the non-terminal (let's call it N1) associated with that line and search it in the left corner table. We find it in the row pertaining to the non-terminal N2. We repeat the same process for N2 and all the resulting non-terminals obtained this way, until we reach non-terminal S. Suppose that S was found through the left corner table value Nk. We take the production "S-> Nk set_of_symbols" and make a top down step adding it to the parse tree. We take the first symbol from the remaining set of symbols, let's call it A and we repeat the following steps:
    a. For the current non-terminal taken from the top-down prediction, we want to find a match with the input string
    b. we take the first word from the remaining input string. We search in the left corner table the non-terminal that contains it in it's table row. We repeat the same process for that non-terminal, by also finding it in a row in the table and noting the non-terminal associated with that row.
    c. We stop when we find through this process the non-terminal A. We then expand A (as we did with S, making a top down prediction), or, if A expands in only one symbol (the one that is already associated with it through the bottom-up steps), we choose the next non-terminal from the top-down

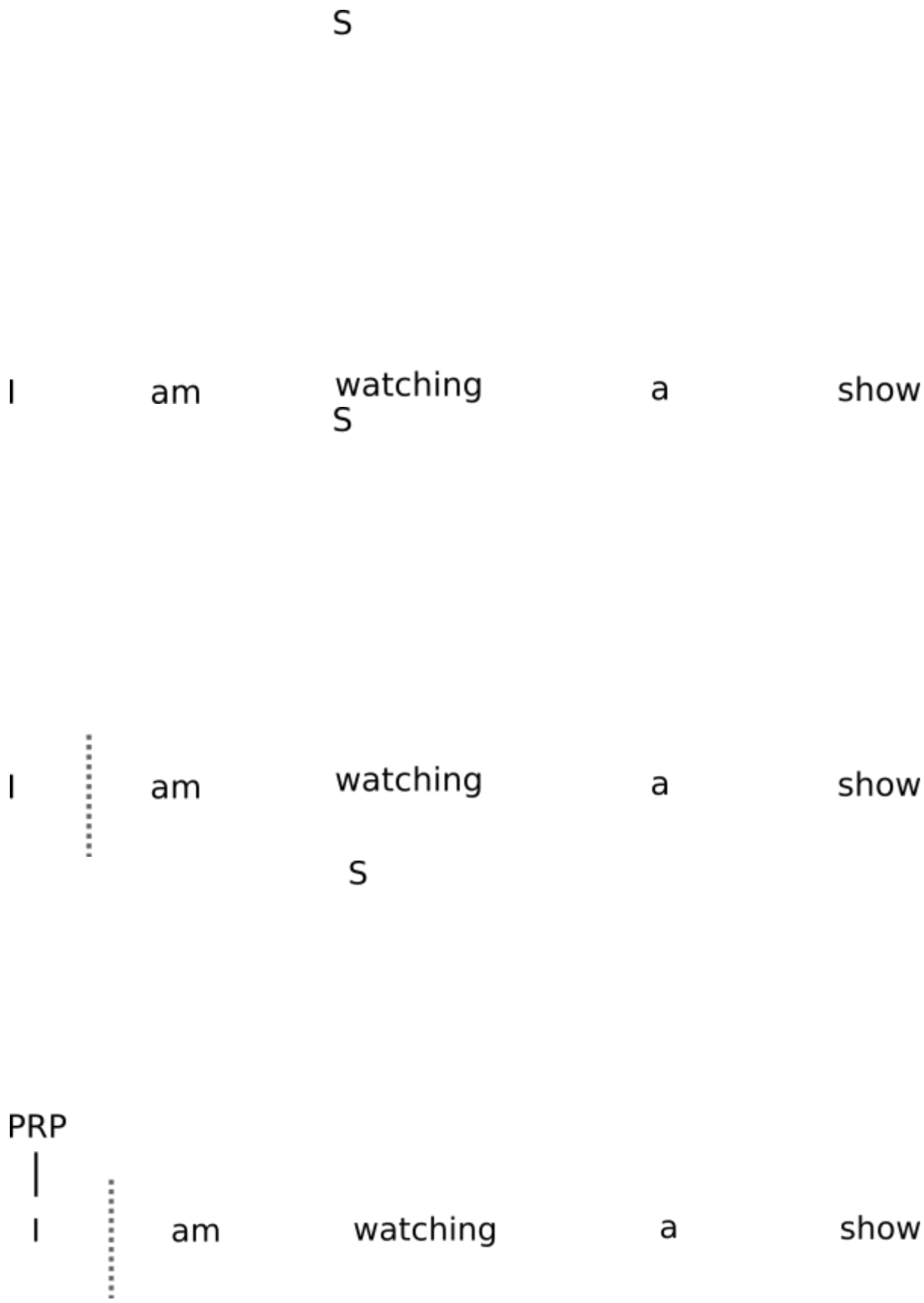
predications as the current non-terminal for which we search a match, and repeat the process

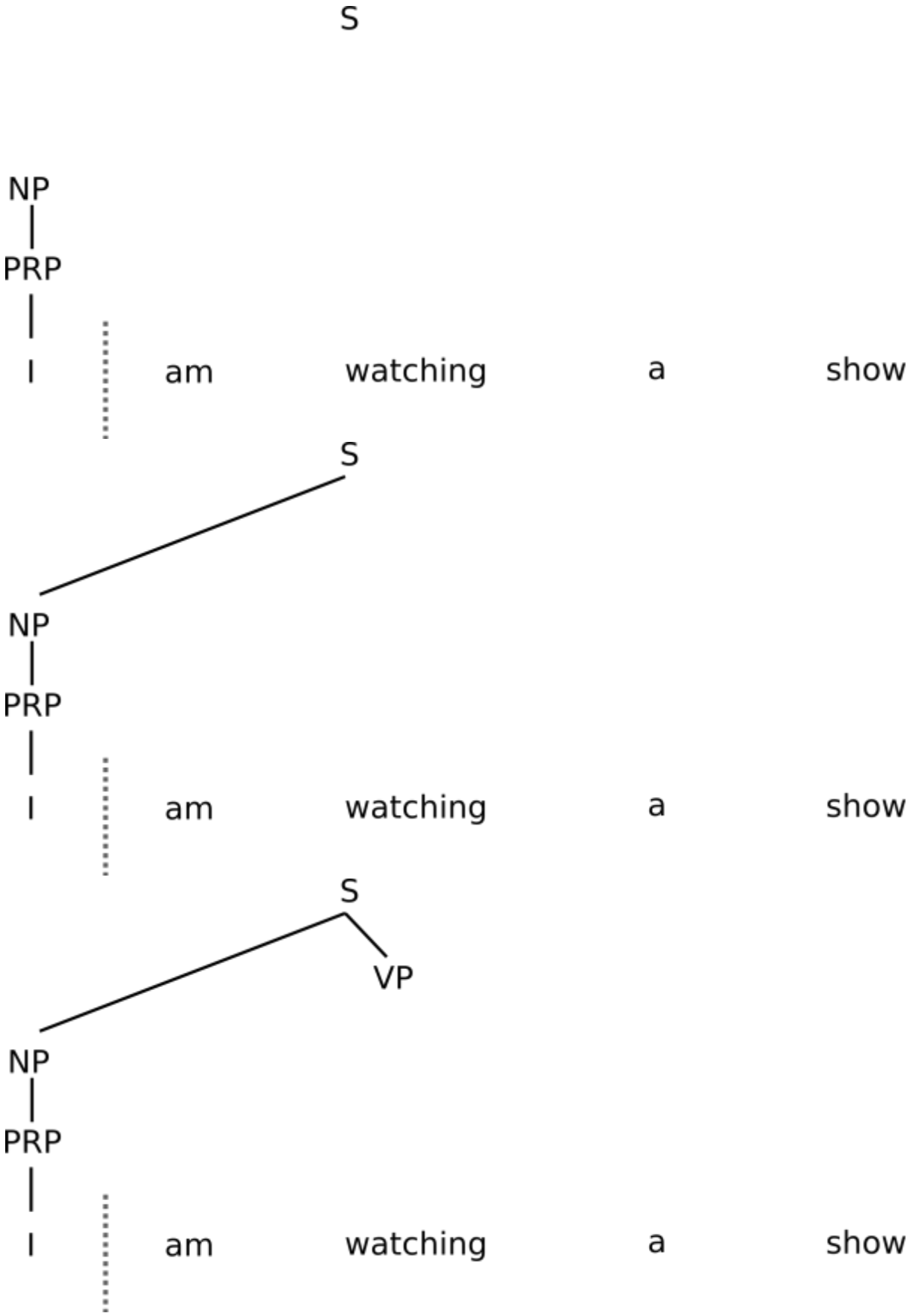
d. In case we don't find a match with the non-terminal A (the bottom-up steps resulted from the search in left corner tables end with another non-terminal, we backtrack (first the bottom-up steps, and if we finish them all we backtrack the top-down prediction as well)

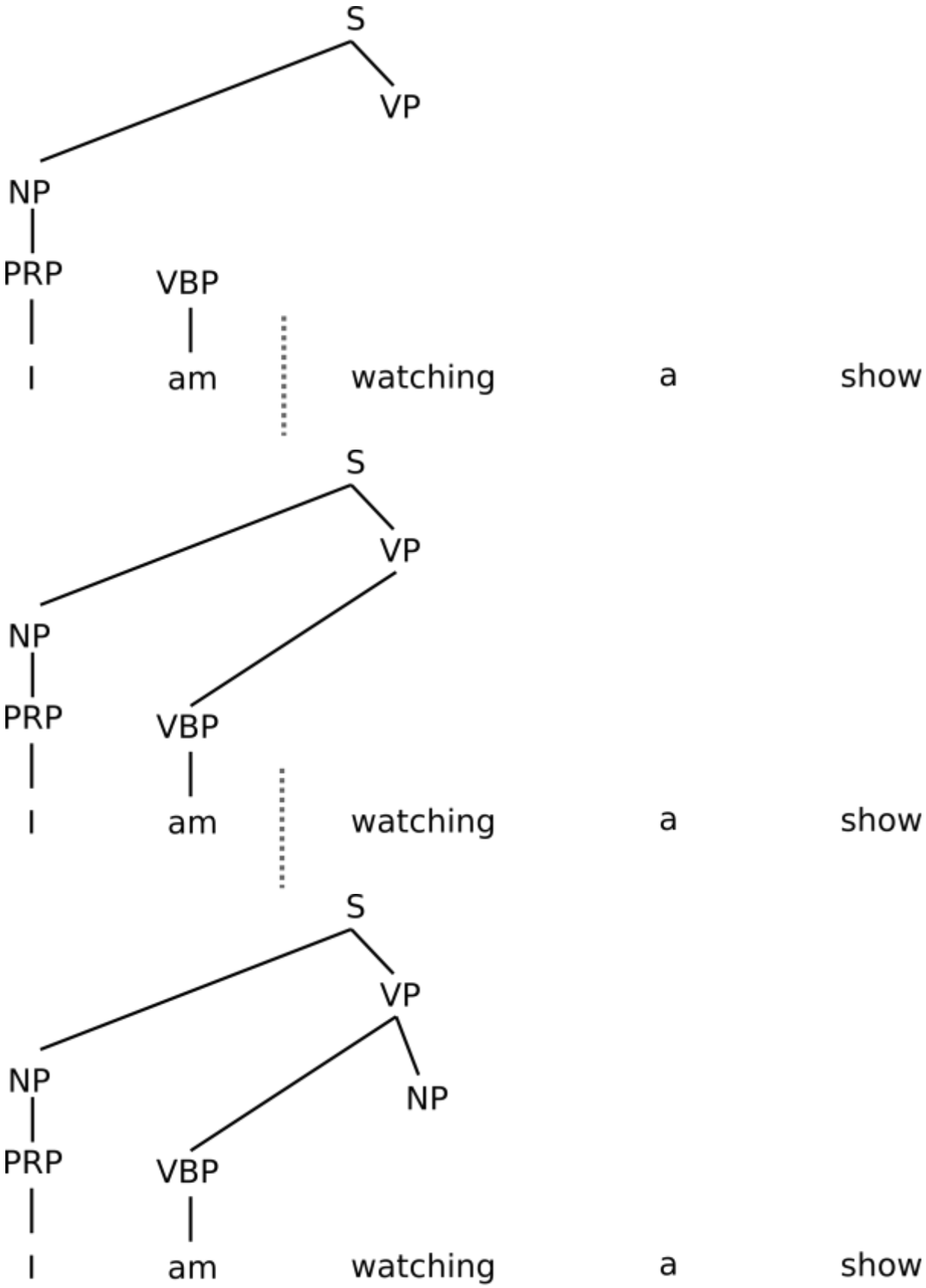
If we consider the grammar above (we also consider that the production rules are processed in this exact order), we have the following steps:

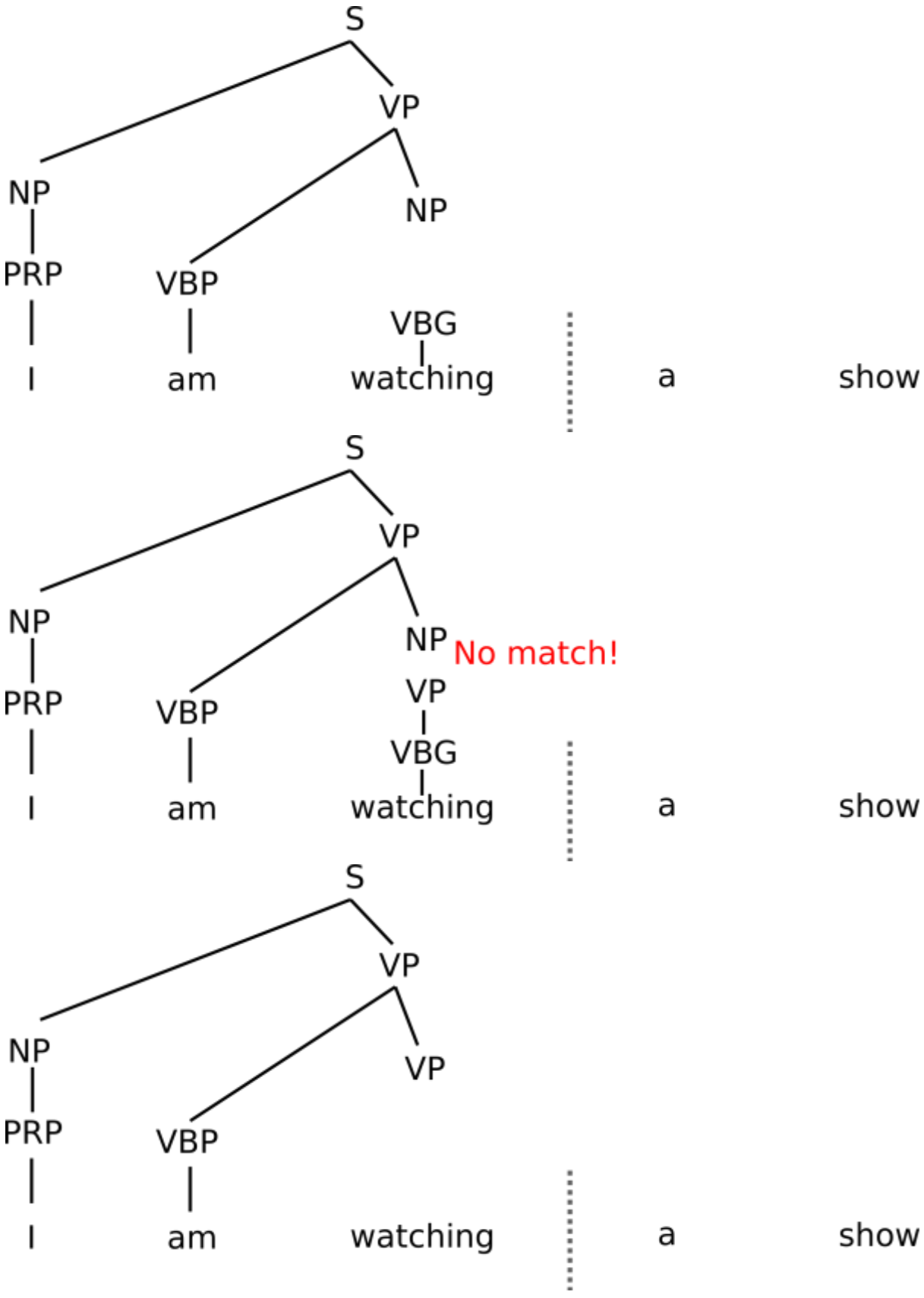
The left corner table would be:

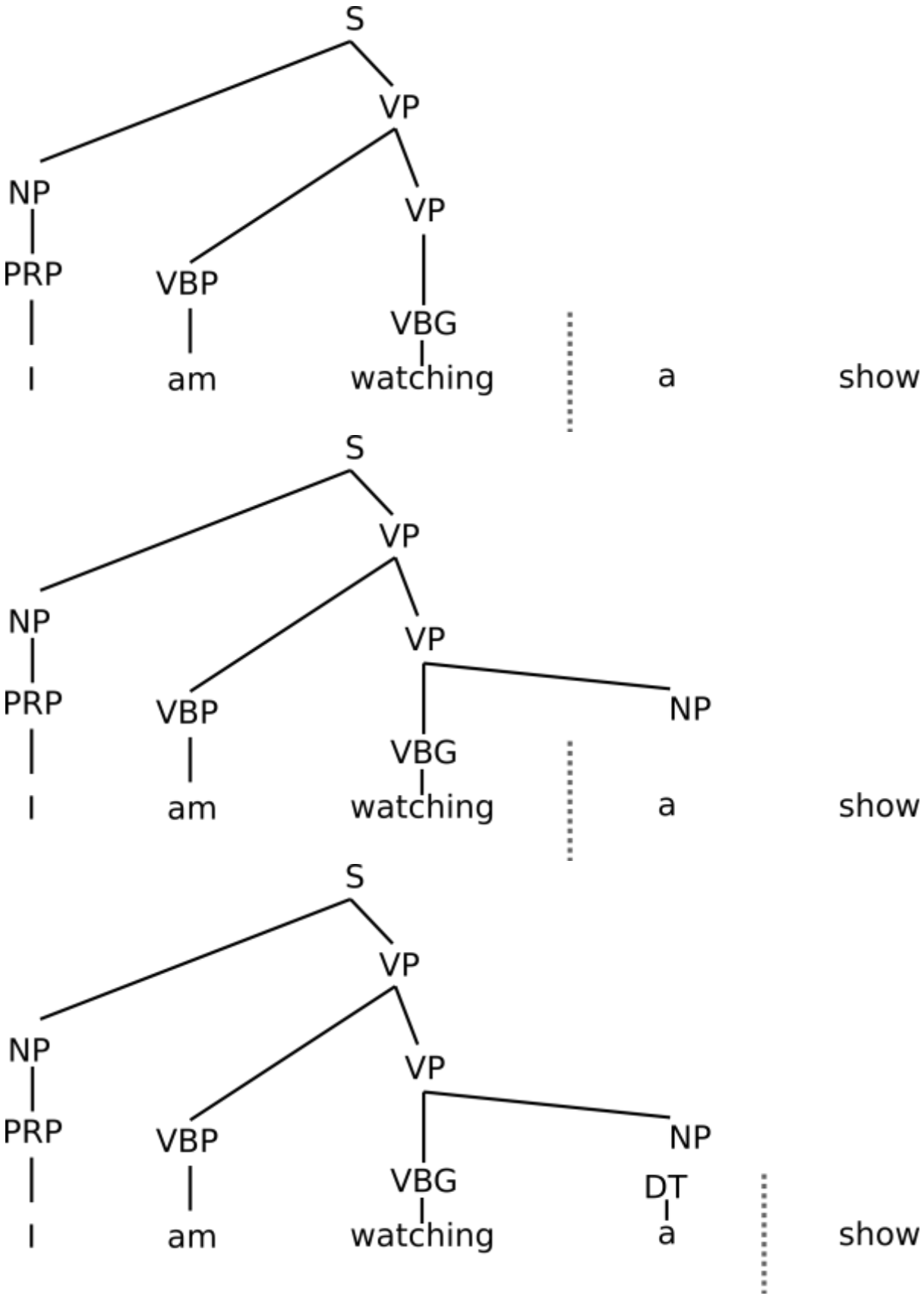
Symbol	**Left corner**
S	NP,VP
NP	DT,PRP
VP	VBP, VBG, TO, VB
NN	"show", "book"
PRP	"I"
VBP	"am"
VBG	"watching"
VB	"show"
DT	"a", "the"
MD	"will"

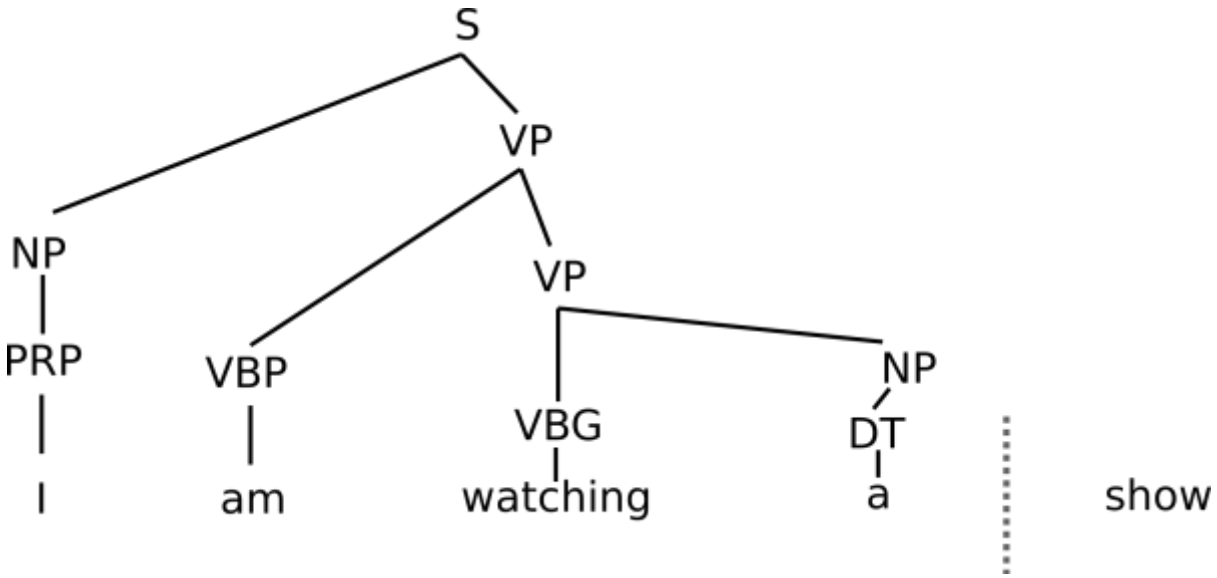

S
I
am
watching
a
show
S
I
am
watching
a
show
S
PRP
I
am
watching
a
show

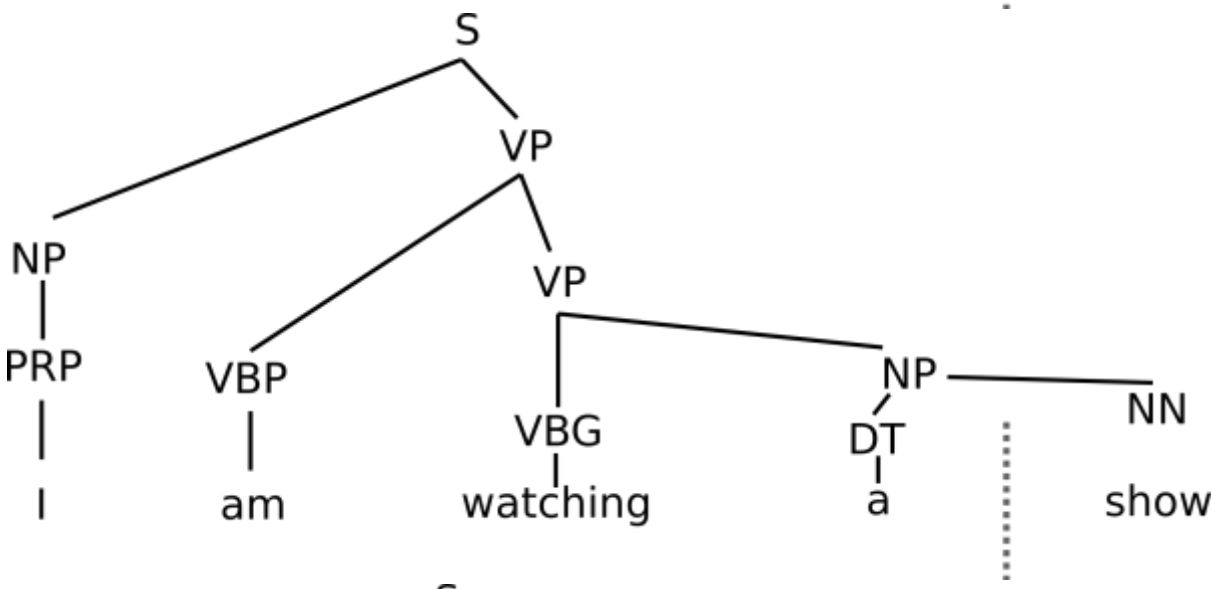

S
NP
PRP
I
am
watching
a
show
S
NP
PRP
I
am
watching
a
show
S
VP
NP
PRP
I
am
watching
a
show


S
VP
NP
PRP
VBP
I
am
watching
a
show
S
VP
NP
PRP
VBP
I
am
watching
a
show
S
VP
NP
NP
PRP
VBP
I
am
watching
a
show


S
VP
NP
NP
PRP
VBP
VBG
I
am
watching
a
show
S
VP
NP
NP
No match!
PRP
VBP
VP
VBG
I
am
watching
a
show
S
VP
NP
VP
PRP
VBP
I
am
watching
a
show


S
VP
NP
VP
PRP
VBP
VBG
I
am
watching
a
show
S
VP
NP
VP
PRP
VBP
NP
VBG
I
am
watching
a
show
S
VP
NP
VP
PRP
VBP
NP
VBG
DT
I
am
watching
a
show





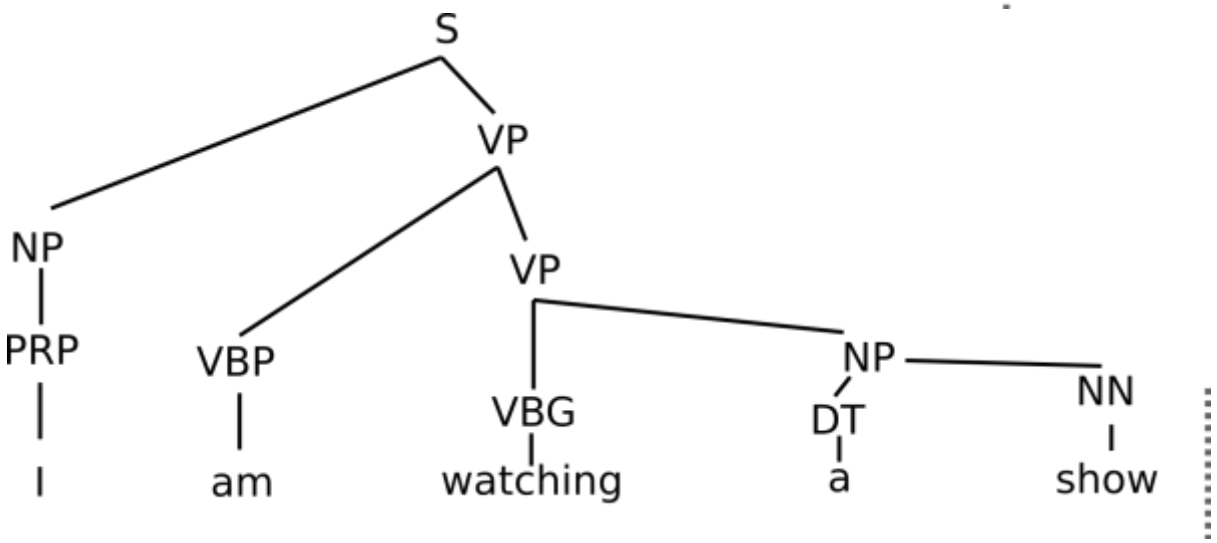


Chart Parser

Well-Formed Substring Tables

It is a type of chart parsing (in Romanian: parser cu agenda). It is used in order to avoid rebuilding subtrees that are already correct. All the previously discussed parsers used backtracking in their algorithm, therefore, we could have subtrees that are processed again because there is a wrong assumption in the subtrees created before them. It's in backtracking's nature to delete(forget) all the computing it made for a wrong prefix in the solution. Each time we build a subtree we save it in a "table" and it will be reused when it's needed. An idea of implementation is given in the NLTK book (Natural Language Processing with Python, by Steven Bird, Ewan Klein, and Edward Loper, 2009).

1. Supposing we have a sentence of n words, we create a matrix of $(n+1)^2$ elements. Let's call this matrix T. The meaning of this matrix is that T[i][j] will contain the root of the subtree containing all the words from i to j-1. At first, the matrix will be empty (initialized with a null value), except elements T[i][i+1] that will contain the i-th word.
2. Next we continuously apply the productions completing the table, until no more changes in the table are made. In order to complete T[i][j] with a label, we must have a number k in [0,n] such that T[i][k] and T[k][j] are both completed (let's say T[i][k] is B and T[k][j] is C) and a production A -> B C. In this case, we'll assign T[i][j]=A. We may have multiple cases for the same line i and column j (from a different reduction of the trees). In this case, we save all the values, so it is better to consider T[i][j] being a list of symbols. An even better representation, in order to easily obtain the responsible productions for the end tree, would be to have the whole production (A -> B C) saved in T[i][j], for example by saving its id (assuming that all the grammar's productions have an id).
3. How do we treat productions with a number of terminals not equal to 2. We may process the grammar and reform the productions, by adding auxiliary ones in order to obtain only two node in each production; for example "A-> B C D" can be changed into "A-> B NewT1" and "NewT1-> C D", where NewT1 is a new terminal used only for this production.
4. The algorithm finishes when no more reductions can be made. If we've obtained S (the sentence node) in T[0][n] we have successfully parsed the sentence.

It is not mandatory to use a matrix, you can save the agenda with a list of subtrees, however you still need to save (in that structure) the indexes for the leafs contained by each subtree.

General lab grading

Exercises proposed during the lab for each lab are worth 1p / 10p, and the deadline is 2 weeks from the lab date (of Grupa 3, last session).

The exercise in lab 1 (on preprocessing) is an exception, no specific deadline, but only serves for additional points to the final grade.

Additionally, you will have larger homework assignments such as the one above, worth 3-4p / 10p, and with longer deadlines specific to each assignment.