Prof. Dr. Andreas Hotho,
M.Sc. Janna Omeliyanenko
Lecture Chair X for Data Science, Universität Würzburg

6. Exercise for "Sprachverarbeitung und Text Mining"

10.12.2021

1 Knowledge Questions

1. Give a definition of Constituency Parsing in your own words.

> Constituency parsing addresses the extraction of a constituency-based parse tree from a sentence, where the sentence is broken down into sub-phrases, also known as constituents, according to a phrase structure grammar.

2. What are possible applications for Constituency Parsing?

> - Recognition or resolution of ambiguity in text
> - Additional information for more complex NLP tasks, e.g.:
>   - Grammar Checking
>   - Question Answering

3. Give a definition of Dependency Parsing in your own words.

> The task of dependency parsing is to extract a dependency parse tree of a sentence. This represents the grammatical structure of the sentence and defines the relationships between "head" words and words that modify these "heads".

4. What are possible applications for Dependency Parsing?

Additional information for more complex NLP tasks, e.g.:

- Grammar checking
- Automatic translation
- Named entity recognition

5. What general (search-based) parsing strategies do you know? Briefly explain how they operate.

In general, a differentiation is made between bottom-up and top-down strategies. Bottom-up algorithms start with words and expand the search space each time, when a rule can be applied to the symbols contained in the input. The process is successful if at the end a symbol S can be generated in at least one state that contains all input symbols (words).
Top-down strategies, on the other hand, generate all states that can be inferred from the grammar, starting at the symbol S. A sentence is successfully parsed if all its tokens can be derived from the grammar.

What are the advantages of each of these strategies?

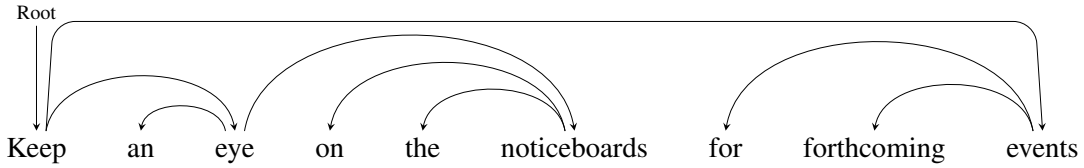
Top-down looks only for trees that can be answers.
Bottom-up builds only trees that match the words.

6. What is the main issue with parsers based on a CFG, regarding the ambiguity of the parsing results? How could it be fixed?

With a CFG only, it is not possible to decide which parse tree is the "best" representation of the sentence. By extending the CFG with probabilities of each rule (PCFG), it is possible to rank the different results.

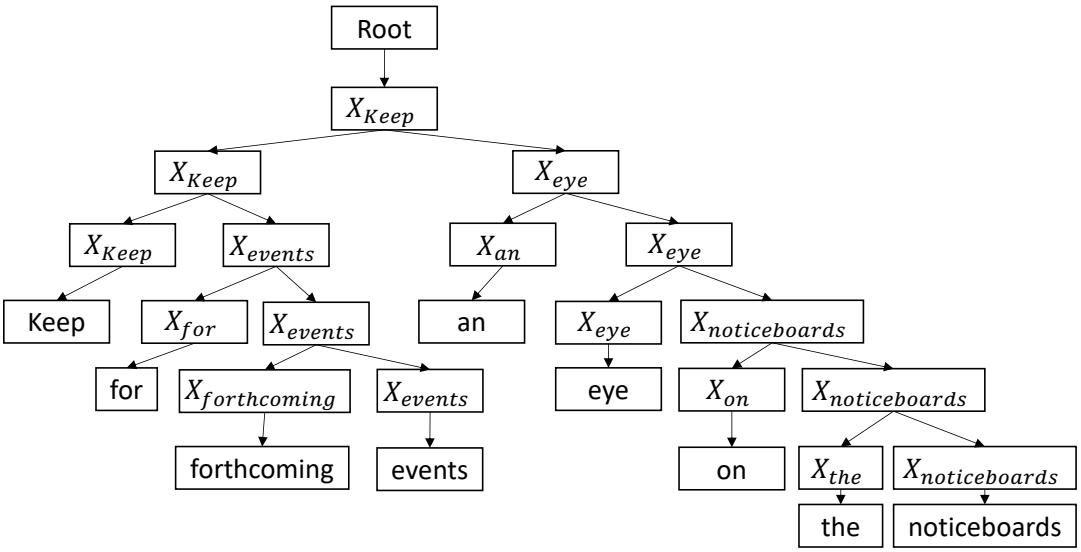
2 Dependency Grammar

Given is the following dependency graph:



Extract a bi-lexical dependency grammar for this dependency tree according to the lecture slides 14-33ff. Visualize your dependency grammar with a tree as done in the lecture slides 14-44.

Grammar:
$Root \rightarrow X_{Keep}$
$X_{Keep} \rightarrow X_{Keep}\, X_{eye}$
$X_{Keep} \rightarrow X_{Keep}\, X_{events}$
$X_{eye} \rightarrow X_{an}\, X_{eye}$
$X_{eye} \rightarrow X_{eye}\, X_{noticeboards}$
$X_{noticeboards} \rightarrow X_{on}\, X_{noticeboards}$
$X_{noticeboards} \rightarrow X_{the}\, X_{noticeboards}$
$X_{events} \rightarrow X_{for}\, X_{events}$
$X_{events} \rightarrow X_{following}\, X_{events}$
$X_{Keep} \rightarrow Keep$
$X_{an} \rightarrow an$
$X_{eye} \rightarrow eye$
$X_{on} \rightarrow on$
$X_{the} \rightarrow the$
$X_{noticeboards} \rightarrow noticeboards$
$X_{for} \rightarrow for$
$X_{following} \rightarrow following$
$X_{events} \rightarrow events$
Possible tree visualizing the grammar:



3 Grammar for Yoda

Create a grammar whose Expressiveness covers at least the following sentences.

```
Much to learn you still have
Always in pairs they are a master and an apprentice
Into exile I must go
```

APPR → *Into*
NN → *exile*
PRO → *I*
V → *must*
V → *go*
S → PP X0
X0 → PRO VP
VP → V V
PP → APPR NN
ADV → *Much*
APPR → *to*
V → *learn*
PRO → *you*
ADV → *still*
V → *have*
VP → APPR V
VP → ADV V
VP → ADV VPS → VP X0

ADV → *Always*
APPR → *in*
ADJ → *pairs*
PRO → *they*
V → *are*
ART → *a*
NN → *master*
KON → *and*
ART → *an*
NN → *apprentice*
PP → ADV PP
PP → APPR ADJ
X1 → KON NP
NP → ART NN
NP → NP X1
VP → V NP
S → PP X0

4 Chomsky-Normalform

A grammar $G = (V, \Sigma, P, S)$ is in the so-called Chomsky Normal Form if every production rule $p \in P$ has one of the following forms:

$$
\begin{aligned}
A &\to BC \\
A &\to a \\
S &\to \epsilon
\end{aligned}
$$

Here Σ denotes the set of all words, V denotes the set of all non-terminal symbols, S denotes the start symbol, and ϵ denotes the empty word. Assume in the following that grammars in language processing can do without the empty word ϵ.

Using the algorithm from the lecture, convert the following grammar (terminal symbols in quotes) into Chomsky Normal Form. It is sufficient to solve each problem once as an example.

$$
\begin{aligned}
S &\to NP\ VP \\
S &\to Aux\ NP\ VP \\
S &\to VP \\
NP &\to Pronoun \\
NP &\to Propernoun \\
NP &\to Det\ Nominal \\
NP &\to \text{"}the\text{"}\ Noun \\
Nominal &\to Noun \\
Nominal &\to Nominal\ Noun \\
Nominal &\to Nominal\ PP \\
VP &\to Verb \\
VP &\to Verb\ NP \\
VP &\to Verb\ NP\ PP \\
VP &\to Verb\ PP \\
VP &\to VP\ PP \\
PP &\to Prep\ NP \\
Verb &\to \text{"}drink\text{"}
\end{aligned}
$$

Algorithm

1. Copy all conform rules into a new grammar
2. Change terminal symbols in 'multi-character' rules to dummy symbols (strategy 2)
3. Convert all unit production rules (strategy 1)
4. Apply strategy 3 to make all rules binary.

Strategies

1. Eliminate all so-called unit production rules ($A \rightarrow B$) by duplicating such a rule so that all possible successors of the symbol B are directly inserted.
2. Introduce a dummy non-terminal symbol and replace the terminal symbol in the rule with it, supplement the set P with a rule that maps the dummy symbol to the terminal symbol.
3. Introduce a dummy symbol that maps 2 of the non-terminal symbols to a dummy symbol. Additionally add a rule to P that maps the dummy symbol to the original non-terminal symbols.

Applying this to the grammar yields:

Step 1: keep the conform rules

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
NP &\rightarrow Det\ Nominal \\
Nominal &\rightarrow Nominal\ Noun \\
Nominal &\rightarrow Nominal\ PP \\
VP &\rightarrow Verb\ NP \\
VP &\rightarrow Verb\ PP \\
VP &\rightarrow VP\ PP \\
PP &\rightarrow Prep\ NP \\
Verb &\rightarrow \text{"}drink\text{"}
\end{aligned}
$$

Step 2: transform terminal symbols in rules to dummy symbols

$$NP \rightarrow \text{"}the\text{"}\ Noun$$
$$NP \rightarrow X_1\ Noun$$
$$X_1 \rightarrow \text{"}the\text{"}$$

Step 3: convert all unit production rules Using $S \rightarrow VP$ as an example. We replace VP with all the possibilities in our grammar.

$$S \rightarrow Verb$$
$$S \rightarrow Verb\ NP$$
$$S \rightarrow Verb\ NP\ PP$$
$$S \rightarrow Verb\ PP$$
$$S \rightarrow VP\ PP$$

Among them, 3 of the 5 newly emerged rules are compliant, we repeat the process for the first emerged rule $S \rightarrow Verb$.

$$S \rightarrow \text{"}drink\text{"}$$

To get the last rule compliant, we need to perform the fourth step of the algorithm.

Step 4: binarizing the rules Using $S \rightarrow Aux\ NP\ VP$ as an example:

$$S \rightarrow X_2\ VP$$
$$X_2 \rightarrow Aux\ NP$$