Faculty of Computer Science, Dalhousie University *23-Nov-2023*

CSCI 4152/6509 — Natural Language Processing

Lecture 22: Natural Language Syntax

Location: Rowe 1011 Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- Neural language model, RNN, stacked and bidirectional RNN
- LSTM, self-attention, transformers
  Part IV: Parsing (Syntactic Processing)
- Prolog introduction
  - unification and backtracking
  - variables, lists; examples: factorial, member

19 Natural Language Syntax

Slide notes:

Natural Language Syntax

- Syntax — NLP level of processing
  - Syntax = sentence structure; i.e., study of the phrase structure
- sýntaxis (Greek) — "setting out together, arrangement"
- Words are not randomly ordered — word order is important and non-trivial
- There are "free-order" languages (e.g., Latin, Russian), but they are not completely order free.
- Reading: Chapter 12 (JM book) or Ch.17 (JM on-line)

Phrase Structure and Dependency Structure

- Two ways of organizing sentence structure:
  - phrase structure
  - dependency structure
- Phrase structure
  - nested consecutive groupings of words
- Dependency structure
  - dependency relations between words
- The main NLP task at the syntax level: parsing
  - given a sentence, find the correct structure

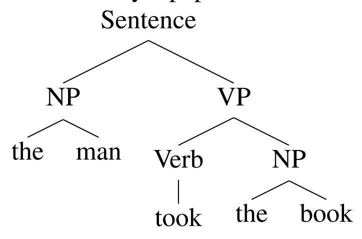
Phrase Structure

- Phrase Structure Grammars or Context-Free Grammars

- A hierarchical view of sentence structure:
  - words form phrases
  - phrases form clauses
  - clauses form sentences
- Parsing: given a sentence find the context-free parse tree; a.k.a. phrase structure parse tree

Phrase Structure Parse Tree Examples

- Phrase Structure parse trees are also called Context-Free parse trees
- This example is from the seminal Noam Chomsky's paper in 1956:

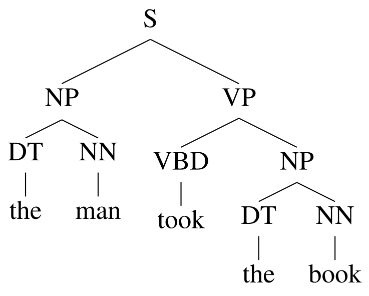


The above example is from the seminal article of Noam Chomsky, "Three models for the description of language" published in IRE Transactions on Information Theory in 1956.

If we follow more closely the Penn treebank tagset, we would rewrite the above parse tree as follows:

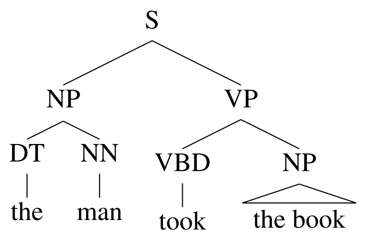
Parse Tree Examples (Penn treebank tagset)

- Using Penn treebank tagset:



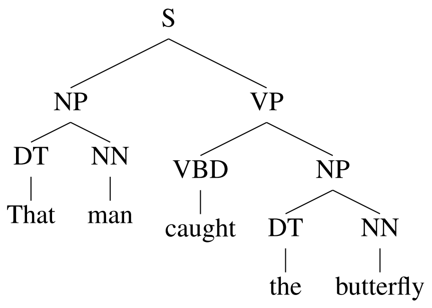
Parse Tree Examples ('triangle' notation)

- Sometimes we simplify a parse tree by ignoring a part of the structure, as in:



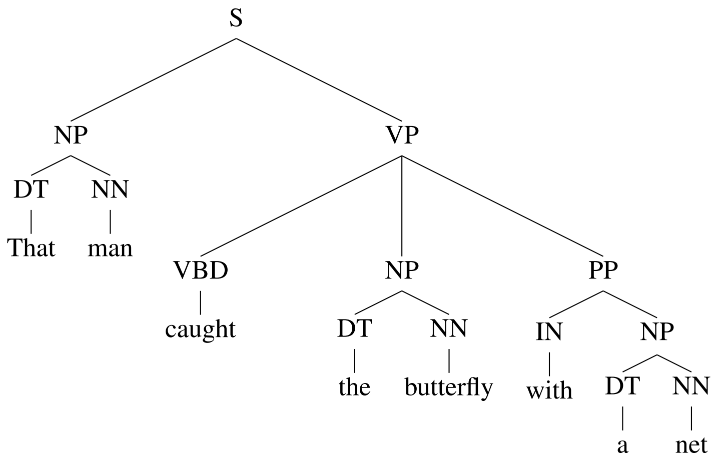
Parse Tree Example 2 ('butterfly')

- Another example:



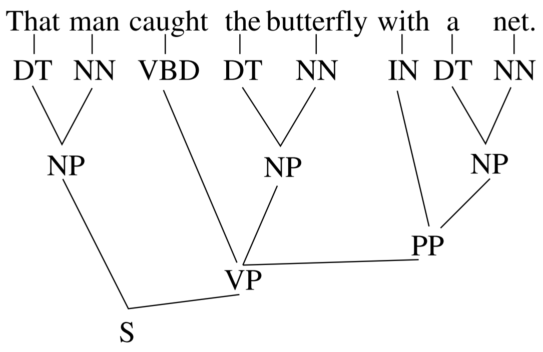
Parse Tree Example3 ('butterfly' extended)

- Extending the previous example:



Parse Tree Example (root bottom)

- Representing parse trees in the bottom-up direction:



Some Basic Notions in Context-Free Trees

- Context-free trees, also called phrase structure trees, parse trees, syntactic trees
- Node relations: root, leaf, parent (mother), child (daughter), sibling, ancestor, descendant, dominate
- Context-free grammar

– Consider for example the context-free grammar induced by the last parse tree shown

20 Context-Free Grammars (CFG) Review

Context-Free Grammars (CFG) Review

CFG is a tuple (V, T, P, S), where

– V is a finite set of **variables** or **non-terminals**; e.g., $V = \{S, NP, DT, NN, VP, VBD, PP, IN\}$
– T is a finite set of **terminals**, words, or lexemes; e.g., $T = \{$That, man, caught, the, butterfly, with, a, net$\}$
– P is a set of **rules** or **productions** in the form $X \to \alpha$, where $X \in V$ and $\alpha \in (V \cup T)^*$; e.g.,
$P = \{S \to NP\ VP,\ \ NP \to DT\ NN,\ \ DT \to \text{That},\ \ NP \to \epsilon\}$
– S is the **start symbol** $S \in V$

Some Notions about CFGs

– CFG, also known as Phrase-Structure Grammar (PSG)
– Equivalent to BNF (Backus-Naur form)
– Idea from Wundt (1900), formally defined by Chomsky (1956) and Backus (1959)
– Typical notation (V, T, P, S); also (N, Σ, R, S)

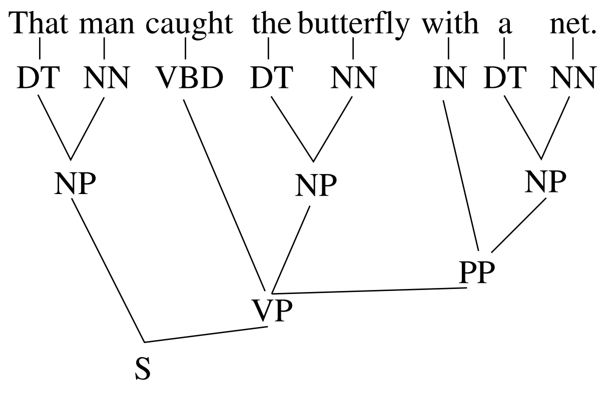
CFG (Context-Free Grammar) is also knows as the Phrase-Structure Grammar (PCG). It is usually referred to as CFG in the Formal Language theory and Computer Science in general, while the term PCG is used in some Computational Linguistic and Linguistic circles. The idea of CFG is traced to Wundt in 1900, but the exact created of the formalism is attributed to Noam Chomsky (1956), who worked in the area of Natural Language Processing, and also to John Backus, who worked in the area of Programming Languages.

CFG Derivations

CFG Derivations

– Direct derivation, derivation
– Example of a direct derivation: $S \Rightarrow NP\ VP$
– Example of a derivation (beginning of):
$S \Rightarrow NP\ VP \Rightarrow DT\ NN\ VP \Rightarrow$ That $NN\ VP \Rightarrow \ldots$
– Left-most and right-most derivation

Parse Tree Example (revisited)



Leftmost Derivation Example

$$\begin{aligned}
S &\Rightarrow NP\ VP \Rightarrow DT\ NN\ VP \Rightarrow \text{That}\ NN\ VP \Rightarrow \text{That man}\ VP \\
&\Rightarrow \text{That man}\ VBD\ NP\ PP \\
&\Rightarrow \text{That man caught}\ NP\ PP \\
&\Rightarrow \text{That man caught}\ DT\ NN\ PP \\
&\Rightarrow \text{That man caught the}\ NN\ PP \\
&\Rightarrow \text{That man caught the butterfly}\ PP \\
&\Rightarrow \text{That man caught the butterfly}\ IN\ NP \\
&\Rightarrow \text{That man caught the butterfly with}\ NP \\
&\Rightarrow \text{That man caught the butterfly with}\ DT\ NN \\
&\Rightarrow \text{That man caught the butterfly with a}\ NN \\
&\Rightarrow \text{That man caught the butterfly with a net}
\end{aligned}$$

Some Notions about CFGs (continued)

- Language generated by a CFG
- Context-Free languages
- Parsing task
- Ambiguous sentences
- Ambiguous grammars
- Inherently ambiguous languages

A language over a set of words (or terminals) in this context (i.e., the formal language context) is any set of strings of words, where a string of words is any finite, possibly empty, sequence of words.

The language generated by a CFG is the set of all strings of words that can be derived from the start symbol using a derivation.

A language is a context-free language, if there is a CFG that generates this language.

The parsing problem for a CFG is the problem of finding all parse trees of an arbitrary string of words, which may include an empty set of trees if the string does not belong to the language generated by the grammar.

Bracket Representation of a Parse Tree

```
(S (NP (DT That)
       (NN man))
   (VP (VBD caught)
       (NP (DT the)
           (NN butterfly))
       (PP (IN with)
           (NP (DT a)
               (NN net)
)  )   )   )
```

Some Notes on CFGs

– Left-hand side (lhs) and right-had side (rhs) of a production

$$\underbrace{S}_{lhs} \rightarrow \underbrace{NP\ VP}_{rhs}$$

– Empty rule (epsilon rule, epsilon production): $V \rightarrow \epsilon$
– Unit production: $A \rightarrow B$, where A and B are non-terminals
– Notational variations:
  – use of ‘|’: $P \rightarrow N \mid A\ P$, instead of $P \rightarrow N,\ \ P \rightarrow A\ P$
  – BNF notation: $P ::= N \mid A\,P$
  – use of word ‘opt’: $NP\ ::=\ DT\ NN\ PP_{opt}$
  – or Kleene star: $NP\ ::=\ DT\ NN\ PP^{*}$

21 Parsing Natural Language in Prolog

21.1 Parsing Natural Language in Prolog using Difference Lists

Prolog’s unique built-in features of backtracking and unification are very convenient for building certain types of parsers for natural languages. To illustrate how this can be done, let us start with a minimal toy example of two sentecnes in English.

Example: Let us consider these twe following two sentences in English: “the dog runs” and “the dogs run”. We will use the following simple CFG to parse these sentences:

```
S  -> NP VP         N  -> dog
NP -> D N           N  -> dogs
D  -> the           VP -> run
                    VP -> runs
```

The grammar is not really correct for English since it allows some non-grammatical sentences to be accepted, such as “the dog run”, but we are going to ignore this for now.

The problem is: How could we build a parser with this grammar to parse a simple sentence such as: “the dog runs”

Using Difference Lists: Idea

Consider rule: `S -> NP VP` and sentence [the,dog,runs]

Using Difference Lists to Parse CFG

The problem of parsing using this grammar can be expressed in the following way in Prolog:

```
s(S,R)  :- np(S,I), vp(I, R).
np(S,R) :- d(S,I), n(I,R).
d([the|R], R).
n([dog|R], R).
n([dogs|R], R).
vp([run|R], R).
vp([runs|R], R).
```

Parsing using Difference Lists

Save this in file `parse.prolog`. On Prolog prompt we type:

```
?- ['parse.prolog'].
% parse.prolog compiled 0.00 sec, 1,888 bytes

Yes
?- s([the,dog,runs],[]).

Yes
?- s([runs,the,dog],[]).

No
```