Faculty of Computer Science, Dalhousie University *25-Nov-2024*

CSCI 4152/6509 — Natural Language Processing

Lecture 20: Syntax of Natural Languages; CYK Algorithm

Location: Carleton Tupper Building Theatre C Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- Bracket representation of a parse tree
- Parsing NL in Prolog using Difference Lists
- Definite Clause Grammar (DCG)
  - Basic DCG example
  - Building a parse tree in DCG
  - Agreement example in DCG
  - Embedded code in DCG
- Probabilistic Context-Free Grammars (PCFG)
- PCFG definition
- PCFG as a probabilistic model
- Typical phrase structure rules in English (started): S

Noun Phrase (NP)

- typically: pronouns, proper nouns, or determiner-nominal construction
- some typical rules

```
NP -> PRP                       e.g.: you
NP -> NNP | NNPS                e.g.: Halifax
NP -> PDT? DT JJ* NN PP*
NP -> NN NN                     e.g.: computer science
```

- in the last rule, we use regular expression notation to describe a set of different rules
- example: all the various flights from Halifax to Toronto
- example 2: all the thick red books on the shelves in the library
- determiners and nominals
- modifiers before head noun and after head noun
- postmodifier phrases

```
NP -> DT JJ* NN RelC
```

Examples of determiners: *a* stop, *the* flights, *this* flight

The determiners can be more complex; e.g., they can consist of a noun phrase and posessive ending *'s*, such as "United's flight" and "Denver's mayor's mother's canceled flight".

However, we will not label those structures as determiners in a context-free parse trees.

Relative Clauses

- RelC — relative clause
- clause (sentence-like phrase) following a noun phrase
- example: gerundive relative clause: flights arriving after 5pm
- example: infinitive relative clause: flights to arrive tomorrow
- example: restrictive relative clause: flight that was canceled yesterday

Verb Phrase (VP)

- organizes arguments around the verb
- typical rules

```
VP -> Verb              intransitive verbs;
                        e.g.: disappear
VP -> Verb NP           transitive verbs:
                        e.g.: prefer a morning flight
VP -> Verb NP NP        ditransitive verbs:
                        e.g.: send me an email
VP -> Verb PP*          sentential complements
VP -> Verb NP PP*
VP -> Verb NP NP PP*
```

- sentential complements, e.g.: You said these were two flights that were the cheapest.

Prepositional Phrase (PP)

- Preposition (IN) relates a noun phrase to other word or phrase
- Prepositional Phrase (PP) consists of a preposition and the noun phrase which is an object of that preposition
- There is typically only one rule for the prepositional phrase:

```
PP -> IN NP
```

- examples: from Halifax, before tomorrow, in the city
- PP-attachment ambiguity

Adjective Phrase (ADJP)

- less common
- examples:
  - She is very sure of herself.
  - . . . the least expensive fare . . .

Adverbial Phrase (ADVP)

- Example:

```
(S (NP preliminary findings)
   (VP were reported
       (ADVP (NP a year) ago)))
```

- another example: years ago

Overview of Syntactic Tags

Tag	Description
S	Sentence
NP	Noun Phrase
VP	Verb Phrase
PP	Prepositional Phrase
ADJP	Adjective Phrase
ADVP	Adverbial Phrase

About Typical Rules

- Only some typical rules are presented
- For example: We see the cat, and you see a dog.
- The sentence could be described with: `S -> S CC S`
- Relative clauses are labeled in Penn treebank using SBAR ($\bar{S}$) non-terminal; e.g.:

```
(S (NP (NP Lorillard Inc.)
       ,
       (NP (NP the unit)
           (PP of (NP (ADJP New York-based)
                      Loews Corp.)))
       (SBAR that
             (S (NP *gap*)
                (VP makes (NP Kent cigarettes))))
       ,)
    (VP stopped (VP using (NP crocidolite))))
```

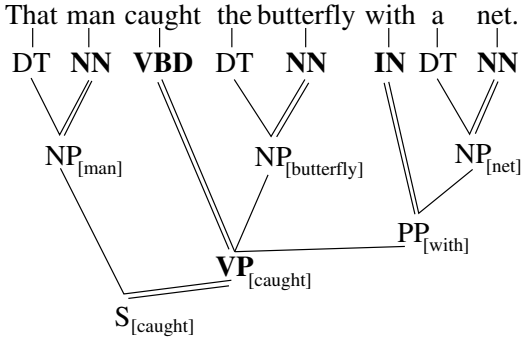
24 Heads and Dependency

Heads and Dependency

- a phrase typically has a central word called head, while other words are direct or indirect dependents
- a head is also called a governor, although sometimes these concepts are considered somewhat different
- phases are usually called by their head; e.g., the head of a noun phrase is a noun

Example with Heads and Dependencies

- the parse tree of "That man caught the butterfly with a net."
- annotate dependencies, head words



Head Feature Principle

The **Head Feature Principle** is the principle that a constituent in a parse tree will have the same set of some characteristic features as its head child. The features that are shared in this way between parent and head child are called **head features**.

The head feature principle is particularly emphasized in some grammar formalisms, such as the Head-driven Phrase Structure Grammar (HPSG).

If we use a context-free grammar, or a similar grammar formalism, to parse natural language, an interesting question is how to detect head-dependent relations. We can achieve it by annotating head child in each context-free rule. Sometimes in literature, these heads are labeled on the right-hand side of each rule using a subscript H, as in:

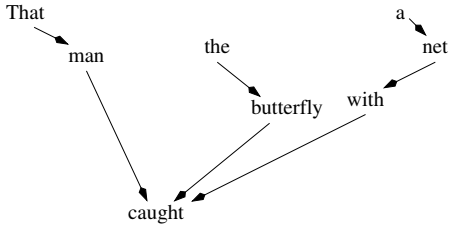
$$NP \rightarrow DT\ NN_H$$

or, using a Head-driven Phrase Structure Grammar (HPSG) style, they are labeled using a letter H before the child constituent, as in:

$$[NP] \rightarrow [DT]\ H[NN]$$

Dependency Tree

- dependency grammar
- example with "That man caught the butterfly with a net."



A dependency tree consists of direct dependence relations between words in a sentences, and the words are the nodes of the tree. A typed dependency tree also includes labels of the edges, describing the types of dependencies. A dependency tree can be easily produced from a context-free parse tree in which the heads of the phrases are labelled.

A dependency grammar is a grammar that defines rules that are used to form a dependency tree.

Arguments and Adjuncts

- There ar two kinds of dependents:
  1. **arguments,** which are required dependents, e.g.,
     We deprived him <u>of food.</u>
  2. **adjuncts,** which are not required;
     - they have a "less tight" link to the head, and
     - can be moved around more easily

     Example:
     We deprived him of food <u>yesterday</u> <u>in the restaurant</u>.

25 CYK Chart Parsing Algorithm

Slide notes:

Efficient Inference in PCFG Model

- Using backtracking is not efficient approach
- Chart parsing is an efficient approach
- We will take a look at the CYK chart parsing algorithm

CYK Chart Parsing Algorithm

- When parsing NLP, there are generally two approaches:
  1. Backtracking to find all parse trees
  2. Chart parsing
- CYK algorithm: a simple chart parsing algorithm
- CYK: Cocke-Younger-Kasami algorithm
- CYK can be applied only to a CNF grammar

The CYK algorithm (Cocke-Younger-Kasami) is a well-known efficient parsing algorithm. The algorithm has a running-time complexity of $O(n^3)$ for a sentence of length n.

CYK can be applied only to a CNF (Chomsky Normal Form) grammar, so if the grammar is not already in CNF, we would have to convert it to CNF. A Context-Free Grammar is in CNF if all its rules are either of the form $A \rightarrow B\,C$, where A, B, and C are nonterminals, or $A \rightarrow w$, where A is a nonterminal and w is a terminal. If a CFG is not in CNF, it can be converted into CNF.

Chomsky Normal Form

- all rules are in one of the forms:
  1. $A \rightarrow B\,C$, where A, B, and C are nonterminals, or
  2. $A \rightarrow w$, where A is a nonterminal and w is a terminal
- If a grammar is not in CNF, it can be converted to it

Is the following grammar in CNF?

S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	N	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

How about this grammar? (Is it in CNF?)

S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	time	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

Note: What if the grammar is not in CNF

There is a standard algorithm for converting arbitrary CFG into CNF. The problem is: How to calculate probabilities of the rules in the new grammar? One way is to sample from the old grammar, and to estimate probabilities in the

new grammar by parsing the sample sentences and counting. The probabilities can also be calculated directly, but it is not a straightforward task.

Here are the steps needed for conversion of an arbitrary CFG into CNF. This is just a partial sketch of the algorithm: calculating probabilities of the new rules in the first two steps is not trivial and it is not given.

Eliminate empty rules $N \to \epsilon$

Find all "nullable" nonterminals, i.e., terminals N such that $N \Rightarrow^* \epsilon$.

From each rule $A \to X_1 \dots X_n$ create new rules by striking out some nullable nonterminals. This is done for all combination of nonterminals in the rule, except for striking out all $X_1 \dots X_n$ if they are all nullable.

Remove empty rules.

If the start symbol is nullable, add $S \to \epsilon$, and treat that as a special case.

Eliminate unit rules $N \to M$

For any two variables A and B, such that $A \Rightarrow^* B$, for all non-unit rules $B \to \zeta$, we add $A \to \zeta$. Remove unit rules.

All possible derivations $A \Rightarrow^* B$ are easy to find since the empty rules are already eliminated.

Eliminate terminals in rules, except $A \to w$

For each terminal w that appears on the right hand side of some rule with some other symbols, we introduce a new nonterminal N_w, and a rule $N_w \to w$ with probability 1. Then we replace w in all other rules with N_w.

Eliminate rules $A \to B_1 B_2 \dots B_n$ **(**$n > 2$**)**

For each rule $A \to B_1 B_2 \dots B_n$ ($n > 2$), we introduce $n-2$ new nonterminals $X_1, \dots, X_{n-2}$, and replace this rule with the following rules: $A \to B_1 X_1$, $X_1 \to B_2 X_2$, $\dots X_{n-2} \to B_{n-1} B_n$, and assign the following probabilities to them: $\mathrm{P}(A \to B_1 X_1) = \mathrm{P}(A \to B_1 B_2 \dots B_n)$, $\mathrm{P}(X_1 \to B_2 X_2) = 1, \dots \mathrm{P}(X_{n-2} \to B_{n-1} B_n) = 1$.

CYK Example

Let us first show the CYK algorithm on an example, before presenting it in detail. We assume that the following grammar in CNF is given:

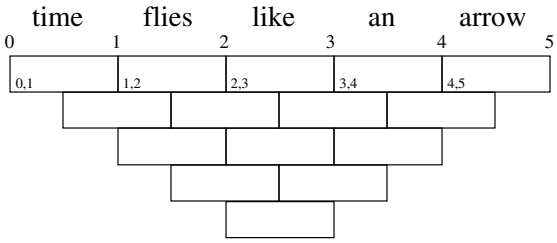
S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	time	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

CYK Example (continued): time flies like an arrow

Using this grammar, we want to parse the sentence 'time flies like an arrow'. We first tokenize the sentence by breaking into the words, and we will enumerate all boundaries between words, starting with the start of the first word in the following way:

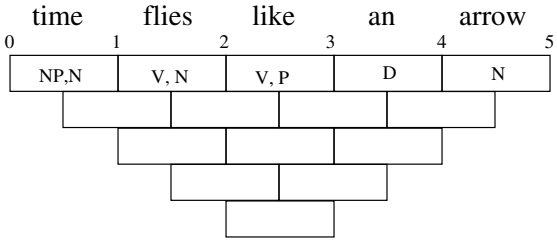
$$_0 \text{ time } _1 \text{ flies } _2 \text{ like } _3 \text{ an } _4 \text{ arrow } _5$$

We organize parsing around a table, called **parsing chart** or simply **chart**, as follows:

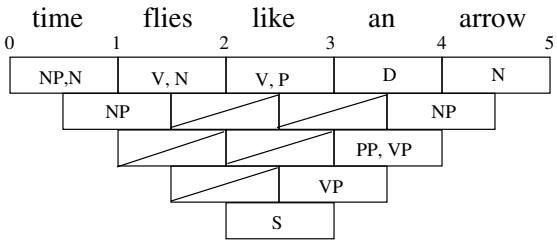


Each entry of the parsing chart is called a **chart entry**. In the empty chart above, we added numbers in the first row of chart entries, which denote spans that are associated with the entries. The first row of chart entries have all spans of lenght 1.

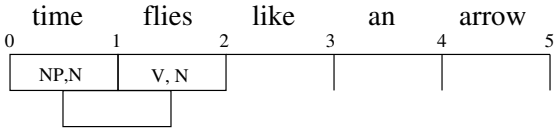
To fill the first row of the table we look at the words and all rules of the form $N \rightarrow w$, where w is the appropriate word, and we add the corresponding non-terminal N to the appropriate chart entry. This is how we fill the first row of the chart and obtain the following:



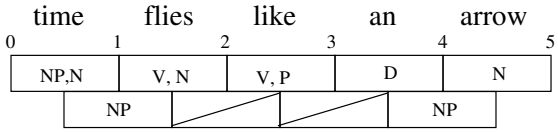
Completely filled chart looks as follows:



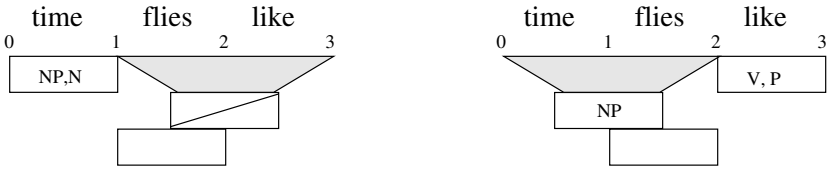
To fill the second row, we look at the span covered by the chart entry. For example, the first chart entry in the second row, covers the span 0–2, as shown below:



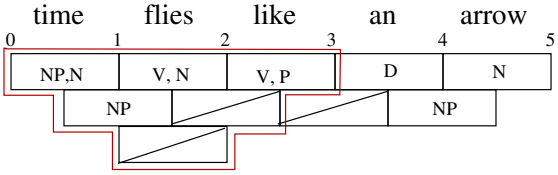
By finding all pairs of non-terminals from the two chart entries that cover sub-spans 0–1 and 1–2, we find pairs:
NP V
NP N
N V
N N
and then we look for the rules, for which one of these sequences is on the right-hand side. We can find the following rule:
NP $\rightarrow$ N N
and we add the non-terminal NP to the entry for the span 0–2. In some cases, we will not find any matching rules and we obtain empty chart entries, which are marked by one crossing diagonal:



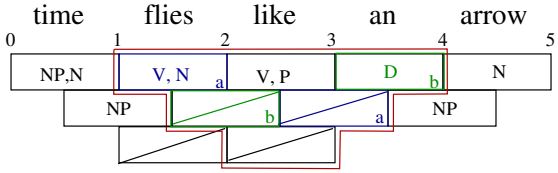
We fill out similarly rows after 2, but it gets a bit more complicated since we will have more than one pair of sub-spans that correspond to a chart entry. For example, the first entry of the third row covers the span 0–3, and then we need to look at the pairs of entries for span 0–1 and 1–3, and 0–2 and 2–3, as shown:



In the first case, when we look at entries ^{0}[NP, N]^{1} and ^{1}[ ]^{3} we do not get any pairs of non terminals. In the second case, we look at the entries ^{0}[NP]^{2} and ^{2}[V, P]^{3} and get the following pairs of non-terminals:
NP V
NP P
for which we cannot find any appropriate rules, whose right-hand sides correspond to these pairs. This is why the entry ^{0}[ ]^{3} remains empty, as follows:



Similarly, we fill the next entry covering the span 1–4 by looking at the pairs of entries covering spans 1–2 and 2–4 (labeled with ‘a’ in the figure below), and the pair 1–3 and 3–4 (labeled with ‘b’):



And we continue the process in this way.