Computational Linguistics

SYNTACTIC PARSING:

Introduction, CYK Algorithm

Martin Rajman

Martin.Rajman@epfl.ch

and

Jean-Cédric Chappelier

Jean-Cedric.Chappelier@epfl.ch

Artificial Intelligence Laboratory

Objectives of this lecture

➥ Introduce syntactic level of NLP

➥ Present its two components: formal grammars and parsing algorithms

Contents

Syntactic level

Analysis of the sentence structure

i.e. "grammatical" analysis (in the linguistic sense)

In **automatic** natural language processing

☞ **formal grammars** and **parsing** theory

two separated/complementary aspects:

procedural	declarative
generic algorithms	data
parsing algorithm	formal grammar

Parsing

Parsing can be seen as:

- RECOGNIZING a sequence of words

  ➥ Is a given sentence correct or not?

or as

- ANALYZING a sequence of words

  ➥ For a syntactically correct sentence, give the set of all its possible interpretations.

  (Returns the empty set for incorrect sentences)

What formalism?

- **symbolic grammars** / statistical grammars
- symbolic grammars:
  - **phrase-structure grammars** (a.k.a *constituency* grammars, *syntagmatic* grammars) recursively decompose sentences into constituants, the atomic parts of which are words ("*terminals*").

    Well suited for ordered languages, not adapted to free-order languages.

    Better expresses structural dependencies.
  - **dependency grammars** focus on words and their *relations* (not necessarly in sequence): functional role of words (rather than categories, e.g. "agent"/"actor" rather than "noun").

    More lexicaly oriented.

    Dependency grammars provide simpler structures (with less nodes, 1 for each word, and less deep), but are less rich than phrase-structure grammars

☞ Modern approach: combine both

Formal phrase-structure grammars

A formal phrase-structure grammar $\mathcal{G}$ is defined by:

- A finite set $\mathcal{C}$ of "non-terminal" symbols — syntactic categories
- A finite set $\mathcal{L}$ of "terminal" symbols — words
- The upper level symbol $S \in \mathcal{C}$ — the "sentence"
- A finite set $\mathcal{R}$ of rewriting rules — syntactic rules

$$\mathcal{R} \subset \mathcal{C}^+ \times (\mathcal{C} \cup \mathcal{L})^*$$

In the NLP field, the following concepts are also introduced:

- lexical rules
- pre-terminal symbols or Part of Speech tags

What kind of grammar for NLP?

Reminder: Chomsky's Hierarchy: complexity is related to the shape of the rules

	language class	grammar type	recognizer	complexity
	regular	$X \rightarrow w$ or $X \rightarrow w\,Y$ (type 3)	FSA	$\mathcal{O}(n)$
embeddings	context-free	$X \rightarrow Y_1 \dots Y_n$ (type 2)	PDA	$\mathcal{O}(n^3)$
crossings	context-dependent	$\alpha \rightarrow \beta \quad \|\alpha\| \leq \|\beta\|$ (type 1)	Turing machine	exp.
	recursively enumerable	$\alpha \rightarrow \beta$ (type 0)	undecidable	

embedding: "The bear the dog belonging to the hunter my wife was a friend of bites howls"

crossing: "Diamonds, emeralds, amethysts are respectively white, green and purple"

What kind of grammar for NLP? (2)

real-life NLP constraints $\Rightarrow$ important limitations on <u>complexity</u>

☞ algorithms at most polynomial time complex

Worst-case complexity of parsing grammar types:

descriptive power

regular and LR(k)	: $\mathcal{O}(n)$	22 ms
context-free	: $\mathcal{O}(n^3)$	11 s
tree-adjoining grammars	: $\mathcal{O}(n^6)$	32 h
more complex models	: exp.	42 days

time constraints

$\Rightarrow$ models actually used: **context-free grammars** (or midly context-sentive grammars)

Notice that in practice, higher level description formalisms might be used for developing the grammars, which are afterwards translated into CFG for practical use (“CF backbone”).

Context Free Grammars

A Context Free Grammar (CFG) $\mathcal{G}$ is (in the NLP framework) defined by:

- a set $\mathcal{C}$ of **syntactic categories** (called ”non-terminals”)
- a set $\mathcal{W}$ of **words** (called ”terminals”)
- an element S of $\mathcal{C}$, called the **top level category**, corresponding to the category identifying complete sentences
- a proper subset $\mathcal{C}_0$ of $\mathcal{C}$, which defines the **morpho-syntactic categories** or “**Part-of-Speech tags**”
- a set $\mathcal{R}$ of rewriting rules, called the **syntactic rules**, of the form:

$$X \rightarrow X_1 \; X_2 \; ...X_n$$

  where $X \in \mathcal{C} - \mathcal{C}_0$ and $X_1...X_n \in \mathcal{C}$
- a set $\mathcal{L}$ of rewriting rules, called the **lexical rules**, of the form:

$$X \rightarrow w$$

  where $X \in \mathcal{C}_0$ and w is a word of the language described by $\mathcal{G}$.

  $\mathcal{L}$ is indeed the **lexicon**

A simplified example of a Context Free Grammar

terminals: a, cat, ate, mouse, the

PoS tags: N, V, Det

non-terminals: S, NP, VP, N, V, Det

rules:

R_1: S$\longrightarrow$ NP VP

R_2: VP $\longrightarrow$ V

R_3: VP $\longrightarrow$ V NP

R_4: NP $\longrightarrow$ Det N

lexicon: N $\longrightarrow$ cat   Det $\longrightarrow$ the   ...

Syntactically Correct

A word sequence is **syntactically correct** (according to $\mathcal{G}$) $\iff$ it can be derived from the upper symbol S of $\mathcal{G}$ in a finite number of rewriting steps corresponding to the application of rules in $\mathcal{G}$.

Notation: $S \Rightarrow^* w_1 \ldots w_n$

Any sequence of rules corresponding to a possible way of deriving a given sentence $W = w_1 ... w_n$ is called a **derivation** of W.

The set (not necessary finite) of syntactically correct sequences (according to $\mathcal{G}$) is by definition the *language* recognized by $\mathcal{G}$

A elementary rewriting step is noted: $\alpha \Rightarrow \beta$; several consecutive rewriting steps: $\alpha \Rightarrow^* \beta$ with α and $\beta \in (\mathcal{C} \cup \mathcal{L})^*$

Example: if as rules we have $X \rightarrow a$, $Y \rightarrow b$ and $Z \rightarrow c$, then for instance:

$X\ Y\ Z \Rightarrow aY Z$ and $X\ Y\ Z \Rightarrow^* abc$

Example

The sequence "*the cat ate a mouse*" is syntactically correct (according to the former example grammar)

$$\begin{array}{rl} & \textit{S} \\ \xrightarrow{R_1} & \textit{NP VP} \\ \xrightarrow{R_4} & \textit{Det N VP} \\ \xrightarrow{L_2} & \textit{the N VP} \\ \xrightarrow{L_1} & \textit{the cat VP} \\ \xrightarrow{R_3} & \textit{the cat V NP} \\ \xrightarrow{L_5} & \textit{the cat ate NP} \\ \xrightarrow{R_4} & \textit{the cat ate Det N} \\ \xrightarrow{L_3} & \textit{the cat ate a N} \\ \xrightarrow{L_4} & \textit{the cat ate a mouse} \end{array}$$

Its derivation is $(R_1, R_4, L_2, L_1, R_3, L_5, R_4, L_3, L_4)$

Example (2)

The sequence "*ate a mouse the cat*" is syntactically *wrong* (according to the former example grammar)

$$
\begin{array}{ll}
 & \textit{S} \\
\stackrel{R1}{\longrightarrow} & \textit{NP VP} \\
\stackrel{R4}{\longrightarrow} & \textit{Det N VP} \\
\stackrel{X}{\longrightarrow} & \text{ate}/\textit{Det N VP}
\end{array}
$$

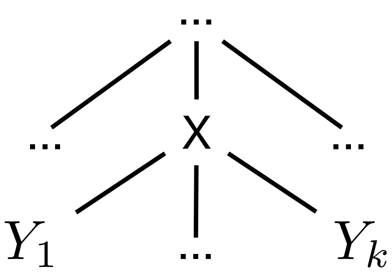
Exercise : *Some colorless green ideas sleep furiously*

Syntactically correct $\neq$ Semantically correct

Syntactic tree(s) associated with a sentence

Each derivation of a sentence W can be represented graphically in the form of a tree in which each rewriting rule is represented as a sub-tree of depth 1: the root (resp. the leaves) corresponds (resp. correspond) to the left-hand side (resp. the right-hand side) of the rule.

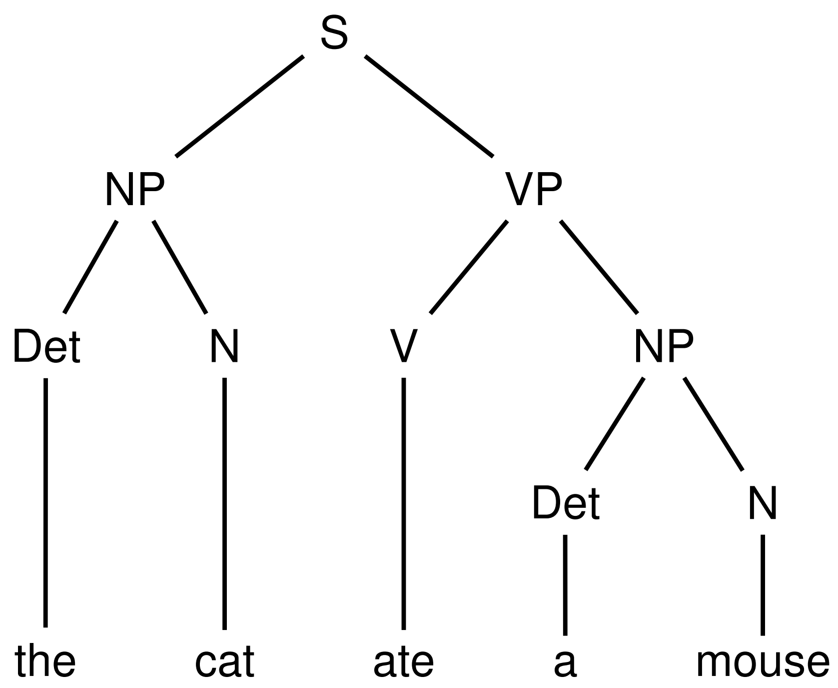
$(\ldots, R_i, \ldots)$ with $R_i : X \rightarrow Y_1 \ldots Y_k \Rightarrow$



Such a tree will be called a **syntactic tree** (or parse tree, or syntactic structure) associated to W by $\mathcal{G}$.

Syntactic tree(s) associated with a sentence

Example:



Mapping between trees and derivations

A priori, several derivations can correspond to the same tree

Example ("the cat ate a mouse"): $R_1, R_4, L_2, L_1, R_3, L_5, R_4, L_3, L_4$ (where the NP is derived before the VP) and $R_1, R_3, L_5, R_4, L_3, L_4, R_4, L_2, L_1$ (where the VP is derived before the NP) correspond to the same tree

However, if, by convention, derivations are restricted to left-most derivations (i.e. derivations where rewriting rules are exclusively applied to the left-most non-terminal), there is a **one-to-one mapping** between derivations and parse trees.

Warning ! This is not true in general for grammars more complex than context-free grammars.

This property is one of the important properties of the CF grammars and will be used for their probabilization.

Syntactic ambiguity

One of the major characteristics of natural languages (in opposition to formal languages) is that they are inherently ambiguous at every level of analysis.

For example, at the syntactic level:

- words are often associated with several parts-of-speech (for example "time" can be a verb or a noun).

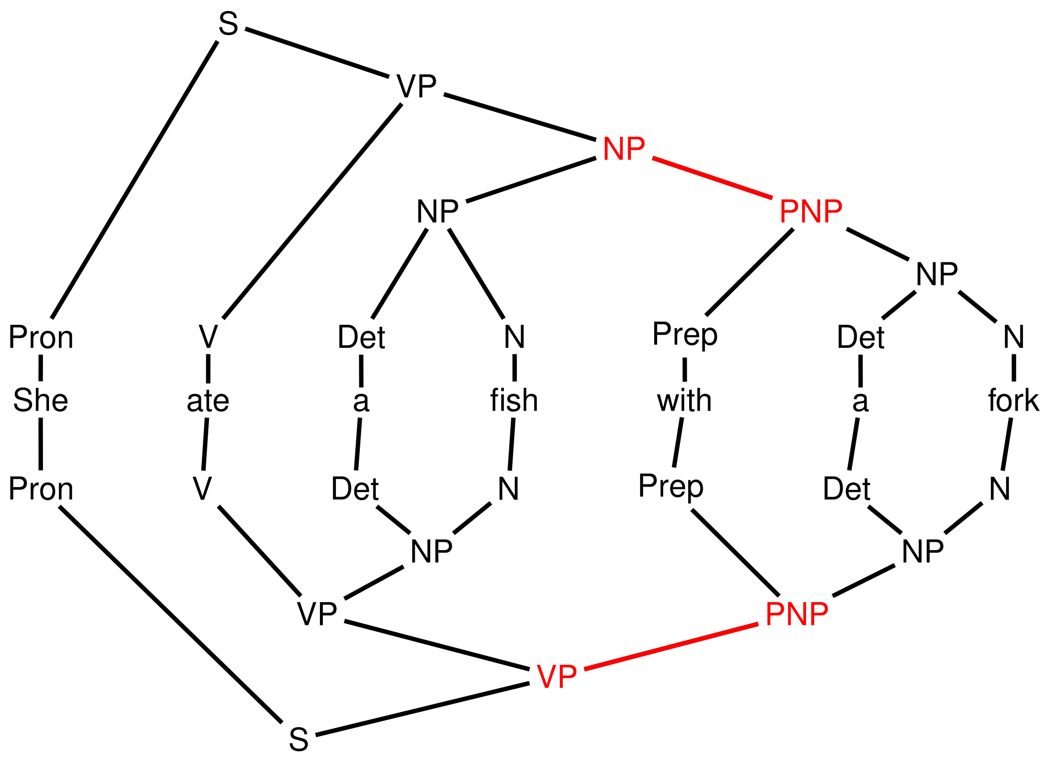
  This can lead to multiple syntactic interpretations corresponding to global structural ambiguities

  Example: time flies like an arrow
- word attachments are often not completely constrained at syntactic level. This can lead to multiple syntactic interpretations corresponding to local structural ambiguities

  Example: She ate a fish with a fork

Examples of syntactic ambiguities

She ate a fish with a fork/bone



Syntactic ambiguity (2)

As the syntactic ambiguity of a given sentence W will be expressed through the association to W of several syntactic structures,

grammars used to describe natural languages **need** to be ambiguous.

This corresponds to a major difference with the grammars that are usually used for formal languages (e.g. programming languages) and have fundamental consequences on the **algorithmic complexity** of the parsers (i.e. syntactic analyzers) that are designed for Natural Language Processing.

Syntactic parsing

One of the main advantages of the CFG formalism is that there exist several **generic parsing algorithms** that can recognize/analyze sentences in a **computationally very efficient** way (low polynomial worst case complexity).

efficient == $\mathcal{O}(n^3)$ worst case complexity

The two most famous of such algorithms are:

- the **CYK** (Cocke-Younger-Kasami) algorithm (first proposed in the early 60's)
- and the **Earley** parser

Input	Output	Resource
sentence	$\left\{ \begin{array}{l} \text{trees (analyser)} \\ \text{yes/no (recognizer)} \end{array} \right.$	CFG

The CYK algorithm

CYK is a bottom-up chart parsing algorithm characterized by 2 interesting features:

- its worst case parsing complexity is $\mathcal{O}(n^3)$ (where n is the number of words of the sentence to be analyzed);
- a very simple algorithm that is easy to implement.

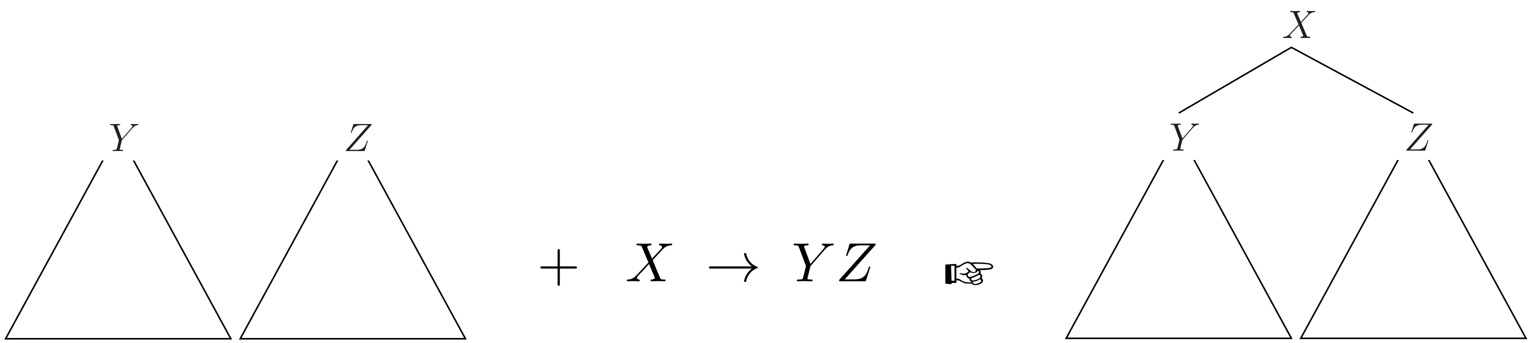
However, its standard implementation suffers from two important drawbacks:

- the CF grammar used by the parser has to be in a predefined format (the Chomsky normal form) and therefore the grammar usually needs to be first converted into this predefined format.
- the complexity is always $\mathcal{O}(n^3)$ even when the grammer is in fact regular

CYK algorithm: basic principles

As it is usual for chart parsing algorithms, the CYK algorithm will compute in an efficient way **all** the possible **syntactic interpretations** of **all the sub-sequences** of the sequence to be analyzed.

Subsequences of the sentences are combined in a bottom-up fashion, using the rules present in the grammar.



How to prevent the space of possible combinations of subsequences from exploding?

➥ Restrict the types of CFG's allowed.

Chomsky Normal Form

Any context-free grammar can be converted into an equivalent **Chomsky Normal Form** (CNF) grammar

A CFG is in CNF if all its syntactic rules are of the form:

$$X \rightarrow X_1\ X_2$$

where $X\ \in\ \mathcal{C} - \mathcal{C}_0$ and $X_1,\ X_2\ \in\ \mathcal{C}$

A context free grammar is in **extended Chomsky Normal Form** if all its syntactic rules are of the form:

$$X \rightarrow X_1\ \ or\ \ X \rightarrow X_1\ X_2$$

where $X\ \in\ \mathcal{C} - \mathcal{C}_0$ and $X_1,\ X_2\ \in\ \mathcal{C}$

Chomsky normal form: example

R1:	*S*	$\rightarrow$	*NP VP*	R1:	*S*	$\rightarrow$	*NP VP*
R2:	*NP*	$\rightarrow$	*Det N*	R2:	*NP*	$\rightarrow$	*Det N*
R3:	*NP*	$\rightarrow$	*Det N PNP*	R3.1:	*NP*	$\rightarrow$	X_1 *PNP*
				R3.2:	X_1	$\rightarrow$	*Det N*
R4:	*PNP*	$\rightarrow$	*Prep NP*	R4:	*PNP*	$\rightarrow$	*Prep NP*
R5:	*VP*	$\rightarrow$	*V*				
R6:	*VP*	$\rightarrow$	*V NP*	R6:	*VP*	$\rightarrow$	*V NP*
R7:	*VP*	$\rightarrow$	*V NP PNP*	R7.1:	*VP*	$\rightarrow$	X_2 *PNP*
				R7.2:	X_2	$\rightarrow$	*V NP*
L5:	*V*	$\rightarrow$	*ate*	L5.1:	*V*	$\rightarrow$	*ate*
				L5.2:	VP	$\rightarrow$	*ate*

☞ increases the number of non-terminals and the number of rules

CYK algorithm: basic principles (2)

The algorithmically efficient organization of the computation is based on the following property:

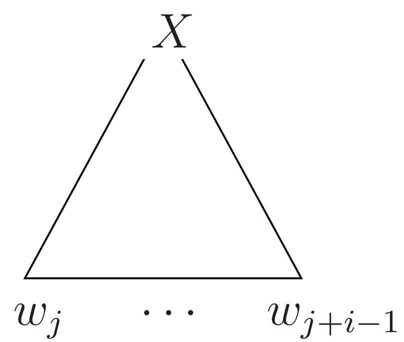
> if the grammar is in Chomsky Normal Form (or in extended Chomsky Normal Form) the computation of the syntactic interpretations of a sequence W of length l only requires the exploration of all the decompositions of W into exactly **two** sub-subsequences, each of them corresponding to a cell in a chart. The number of pairs of sub-sequences to explore to compute the interpretations of W is therefore $l-1$.

Idea: put all the analyses of sub-sequences in a chart

CYK algorithm: basic principles (3)

The syntactic analysis of an n-word sequence $W = w_1 ... w_n$ is organized into a half-pyramidal table (or chart) of cells $C_{i,j}$ ($1 \leq i \leq n,\ 1 \leq j \leq n$), where the cell $C_{i,j}$ contains all the possible syntactic interpretations of the sub-sequence $w_j ... w_{j+i-1}$ of i words starting with the j-th word in W.

$X \in C_{ij}$

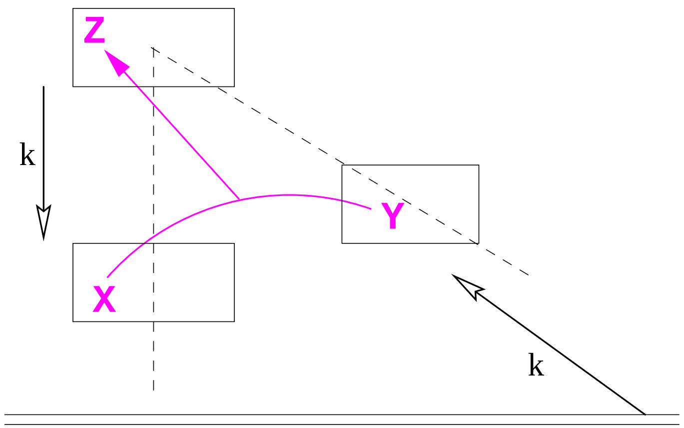


The computation of the syntactic interpretations proceeds row-wise upwards (i.e. with increasing values of i).

CYK Algorithm: principle

8	S							
7								
6			VP, X_2					
5	S			NP				
4								
3	S		VP, X_2			PNP		
2	NP, X_1			NP, X_1			NP, X_1	
1	Det	N	V, VP	Det	N	Prep	Det	N
l/i	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

Formal algorithm



for all $2 \leq i \leq n$ (row) **do**
  for all $1 \leq j \leq n - i + 1$ (column) **do**
    for all $1 \leq k \leq i - 1$ (decomposition) **do**
      for all $X \in$ chart$[i-k][j]$ **do**
        for all $Y \in$ chart$[k][i+j-k]$ **do**
          for all $Z \to X\ Y \in \mathcal{R}$ **do**
            Add Z to chart$[i][j]$

Analyzer or recognizer?

- The preceeding algorithm does not store the parse trees.

  ➥ Recognizer (check wheter S is in top cell or not) or, for an analyser, need to reconstruct the parse trees.
- For an analyzer, it's definitely better to store the parse trees in the chart while parsing:

  Extend

  Add Z to chart$[i][j]$

  with

  Add pointers to X and Y to the interpretations of Z in chart$[i][j]$

CYK algorithm: worst case complexity

As the computation of the syntactic interpretations of a cell $C_{i,j}$ requires $(i-1)$ explorations of pairs of cells $(1 \leq k \leq i-1)$, the total number of explorations is therefore

$$\sum_{i=2}^{n} \sum_{j=1}^{n-i+1} (i-1) = \sum_{i=1}^{n} (n-i+1).(i-1) = \mathcal{O}(n^3)$$

A cell contains at most as many interpretations as the number $|\mathcal{C}|$ of syntactic categories contained in the grammar, the worst case cost of an exploration of a pair of cells corresponds therefore to $|\mathcal{C}|^2$ accesses to the grammar.

Complexity (2)

As cost of the access to the rules in the grammar can be made constant if efficient access techniques (based on hash-tables for example) are used, the worst case computational complexity of the analysis of a sequence of length n is:

$\mathcal{O}(n^3)$ and $\mathcal{O}(|\mathcal{C}|^2)$

We can here see one drawback of the CNF: $\mathcal{C}$ is increased.

We later present a modified version of the CYK algorithm where CNF is no longer required ($\mathcal{C}$ is then smaller)

Notice: Once the chart has been filled ($O(n^3)$ complex), **one** parse tree of the input sentence can be extracted in $O(n)$.

Complexity (3)

PITFALL!! It is easy to implement this algorithm in such a way that the complexity becomes $\mathcal{O}(\exp n)$!

If indeed the non-terminals produced in a cell are **duplicated** (instead of **factorizing** their interpretations), their number can become exponential!

Example: `S -> S S` `S -> a`

S S S S S			
S S	S S		
S	S	S	
S	S	S	S
a	a	a	a

EXPONENTIAL

S: •, •, •			
S: •, •	S: •, •		
S: •	S: •	S: •	
S: •	S: •	S: •	S: •
a	a	a	a

CUBIC

Keypoints

- Role of syntactic analysis is to recognize a sentence and to produce its structure
- Different types of formal grammars, relation between description power and time constraints
- CYK algorithm, its principles and complexity

References

[1] D. Jurafsky & J. H. Martin, *Speech and Language Processing*, chap. 12, 13, and 16, Prentice Hall, 2008 (2nd ed.).

[2] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, chap. 3, MIT Press, 2000

[3] N. Indurkhya and F. J. Damerau editors, *Handbook of Natural Language Processing*, chap. 4, CRC Press, 2010 (2nd edition)

[4] J.-M. Pierrel ed., *Ingénierie des langues*, chap. 2, Hermes, 2000.