Syntactic Parsing: Introduction, CYK Algorithm

M. Rajman & J.-C. Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectives of this lecture

- ➥ Introduce syntactic level of NLP
- ➥ Present its two components: formal grammars and parsing algorithms

Contents:

M. Rajman & J.-C. Chappelier

Syntactic level

Analysis of sentence structure

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

In **automatic** natural language processing ... this requires ...

☞ **formal grammars** and **parsing** (algorithms)

two separated/complementary aspects:

procedural generic algorithms	declarative data
parsing algorithm	formal grammar

M. Rajman & J.-C. Chappelier

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... i.e. associated structures ...
  (Returns the empty set for incorrect sentences)

M. Rajman & J.-C. Chappelier

Syntactic constraints : what is a "correct" word sequence ?...

Let's first play a game...

Consider the following multi-set of 14 words:

{cat, couch, he, lovely, nice, neighbor, of, on, sat, talked, the, the, the, with}

From such a multi-set, one can derive 14! = 87'178'291'200 (!!) possible sequences...

...most of which do not correspond to any reasonably acceptable sentence :

- cat couch he lovely nice neighbor of on sat talked the the the with
- he cat the nice lovely the neighbor sat of talked on with the couch
- ...

But some do! . . .

M. Rajman & J.-C. Chappelier

Some possible sentences

Here are some:

- the lovely cat of the neighbor he talked with sat on the nice couch
- the nice neighbor he sat with talked of the cat on the lovely couch
- the neighbor he sat with talked lovely of the cat on the nice couch
- the neighbor he sat on talked with the nice couch of the lovely cat

M. Rajman & J.-C. Chappelier

What is acceptable and what is not?

A sequence of words can be rejected for several different reasons:

- the words are not in the “right” order:

  cat the on sat the couch nice

  - ☞ the rules defining what are the acceptable word orders in a given language are called “***positional constraints***”
- related word pairs are not matching “right”:

  cats eats mice

  - ☞ the rules defining what are the acceptable word pairs in a given language are called “***selectional constraints***” (“*agreement rules*”)

M. Rajman & J.-C. Chappelier
EPFL

What is acceptable and what is not? (2)

It is not enough for a sequence of words to satisfy all positional and selectional constraints to be acceptable,
see Chomsky's famous example:

Colorless green ideas sleep furiously.

but the reason is different: the sequence is rejected because it is meaningless; indeed, how can something colorless be green ?
or a sleep to be furious ?

As this type of problem is related to meaning, it will not be considered here;
we will consider any sequence satisfying all **positional** and **selectional** constraints as acceptable;

to avoid potential confusion, we will refer to such sequences as "***syntactically acceptable***".

M. Rajman & J.-C. Chappelier
EPFL

Where is the border?

- Syntactic acceptability is not as clear cut as one may think!
- The underlying hypothesis is that any syntactically acceptable sequence may possibly be given a meaning, even if this may require some context to guarantee that a large enough fraction of speakers indeed understand it as intended (which is crucial for any linguistic entity to be truly useful, but, maybe, in pure poetry)
- For example: What do you understand if one talks about a "*small giant*"?...

... while, by definition, a "giant" cannot be "small"...

M. Rajman & J.-C. Chappelier

Where is the border? (2)

- Now, what do you understand, if "*small giant*" is included in the following context:
  "The sheer size of a company does not guarantee its survival; it must also remain agile to adapt to rapidly changing economic conditions. As soon as a large company begins to be hampered by heavy internal procedures, it gradually turns into a small giant, and represents an easy prey for its competitors."
- However, the situation may become fuzzier, if the required context gets harder to create:
  "*giving something to someone*" is clear,
  "*giving something for someone*" as well,
  but how should we interpret "*giving something beyond someone*" ?

... and, what about the 4th of the "possible sentences" given earlier? ...

M. Rajman & J.-C. Chappelier
EPFL

Positional constraints

As already mentioned, positional constraints govern the word order in a language:

the more such constraints, the more the language tends to be fixed order (e.g. French, German),

the less, the more it tends to be free order (e.g. Latin, Italian)

For example: in English "*girls like roses*" is acceptable,

while "*girls roses like*" or "*like girls roses*" are not

(and "*roses like girls*" is acceptable, but means something else);

in Latin, virtually any combination of "*puellae rosas amant*" is acceptable and means the same (up to, possibly, a different emphasis)

How to deal with selectional constraints?

As already mentioned, selectional constraints are taking into account constraints such as agreement rules that are further restricting the word sequences to be considered as (syntactically) acceptable

For example, in English “*cats eat mice*” is acceptable, while “*cats eats mice*” is not, because the number agreement between “*cats*” (plural) and “*eats*” (singular) is violated.

Agreement rules can be taken into account by preserving the required morpho-syntactic features in the PoS tags assigned to words (e.g. a number agreement will require to use PoS tags such as `NOUNs` (noun singular), `NOUNp` (noun plural), `VERBs` (verb singular), and `VERBp` (verb plural).

What formalism?

- **symbolic grammars** / statistical grammars
- symbolic grammars:
  - **phrase-structure grammars** (a.k.a *constituency* grammars, *syntagmatic* grammars) recursively decompose sentences into constituents, the atomic parts of which are words ("*terminals*").
    Well suited for ordered languages, not adapted to free-order languages.
    Better express structural dependencies. (typically positional constraints)
    constituent = sequence of "non terminal" syntactic categories
  - **dependency grammars** focus on words and their *relations*
    Describe functional dependencies between words (e.g. subject-verb relation)
    More lexically oriented.
    Dependency grammars provide simpler structures (with less nodes, 1 for each word, and less deep), less rich than phrase-structure grammars
    Better express relations (typically selectional constraints)

☞ Modern approach: combine both

M. Rajman & J.-C. Chappelier
EPFL

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

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

Example of rewriting rule:

if $X_1 X_2 Y_1 Y_2 Z \in \mathcal{C}$ and $w \in \mathcal{L}$

then $(X_1 Z X_2, w Y_1 Y_2)$ in $\mathcal{R}$ means that the sequence $X_1 Z X_2$ can be rewritten into $w Y_1 Y_2$ and the rule is usually written as:

$$X_1 Z X_2 \rightarrow w Y_1 Y_2$$

M. Rajman & J.-C. Chappelier

What kind of grammar for NLP?

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

expressive power ↓

	Grammar type	Rule shape	recognizer	complexity
	regular	$X \rightarrow w$ or $X \rightarrow w\,Y$ (type 3)	FSA	$\mathcal{O}(n)$
embeddings	context-free	$X \rightarrow Y_1 \ldots Y_n$ (type 2)	PDA	$\mathcal{O}(n^3)$
crossings	context-dependent	$\alpha \rightarrow \beta \;\; \lvert\alpha\rvert \leq \lvert\beta\rvert$ (type 1)	Turing machine	exp.
	recursively enumerable	$\alpha \rightarrow \beta$ (type 0)	undecidable	

algorithmic efficiency ↑

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 ⇒ important limitations on complexity
☞ algorithms at most polynomial time complex

Worst-case complexity of parsing grammar types:

expressive 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

↑ real time

the right tradeoff between expressive power and algorithmic complexity must be found!...

⇒ models actually used: **context-free grammars** (or mildly context-sensitive 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** ("non-terminals")
- a set $\mathcal{L}$ of **words** ("terminals")
- an element S of $\mathcal{C}$, called the **top level category**, corresponding to the category identifying complete sentences
- a proper subset $\mathcal{T}$ of $\mathcal{C}$, which defines the **morpho-syntactic categories** or "**Part-of-Speech tags**" (called "pre-terminals")
- a set $\mathcal{R}$ of rewriting rules, called the **syntactic rules**, of the form:

  $$X \rightarrow X_1\ X_2\ \ldots X_n$$

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

  $$X \rightarrow w$$

  where $X \in \mathcal{T}$ 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:

syntactic rules

R_1: S → NP VP

R_2: VP → V

R_3: VP → V NP

R_4: NP → Det N

(lexicon) lexical rules:

N → cat (L_1)

Det → the (L_2)

V → ate (L_5)

N → mouse (L_4)

Det → a (L_3)

M. Rajman & J.-C. Chappelier
EPFL

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: $S \Rightarrow^* w_1 \dots w_n$

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

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

An 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:

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

Example : Let us show that...

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

We have :

("⟹" is written "$\xrightarrow{R}$", where R identifies the rule used for the rewriting step)

$$
\begin{array}{ll}
 & S \\
(1) \xrightarrow{R_1} & NP\ VP \\
(2) \xrightarrow{R_4} & Det\ N\ VP \\
(3) \xrightarrow{L_2} & the\ N\ VP \\
(4) \xrightarrow{L_1} & the\ cat\ VP \\
(5) \xrightarrow{R_3} & the\ cat\ V\ NP \\
(6) \xrightarrow{L_5} & the\ cat\ ate\ NP \\
(7) \xrightarrow{R_4} & the\ cat\ ate\ Det\ N \\
(8) \xrightarrow{L_3} & the\ cat\ ate\ a\ N \\
(9) \xrightarrow{L_4} & the\ cat\ ate\ a\ mouse
\end{array}
$$

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

Example (2)

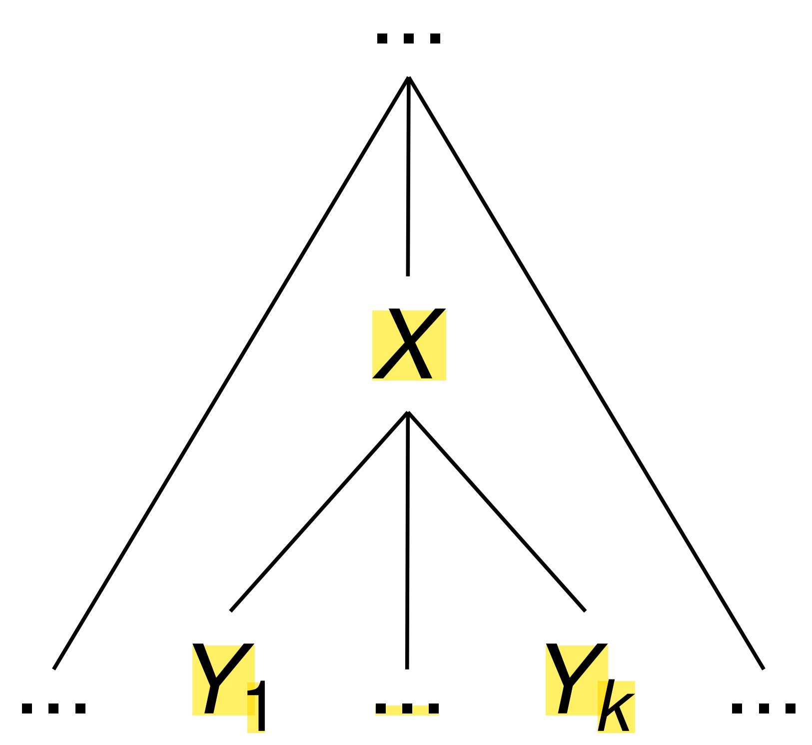
The sequence "*ate a mouse the cat*" is syntactically *wrong* (according to the former example grammar)

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

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.

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

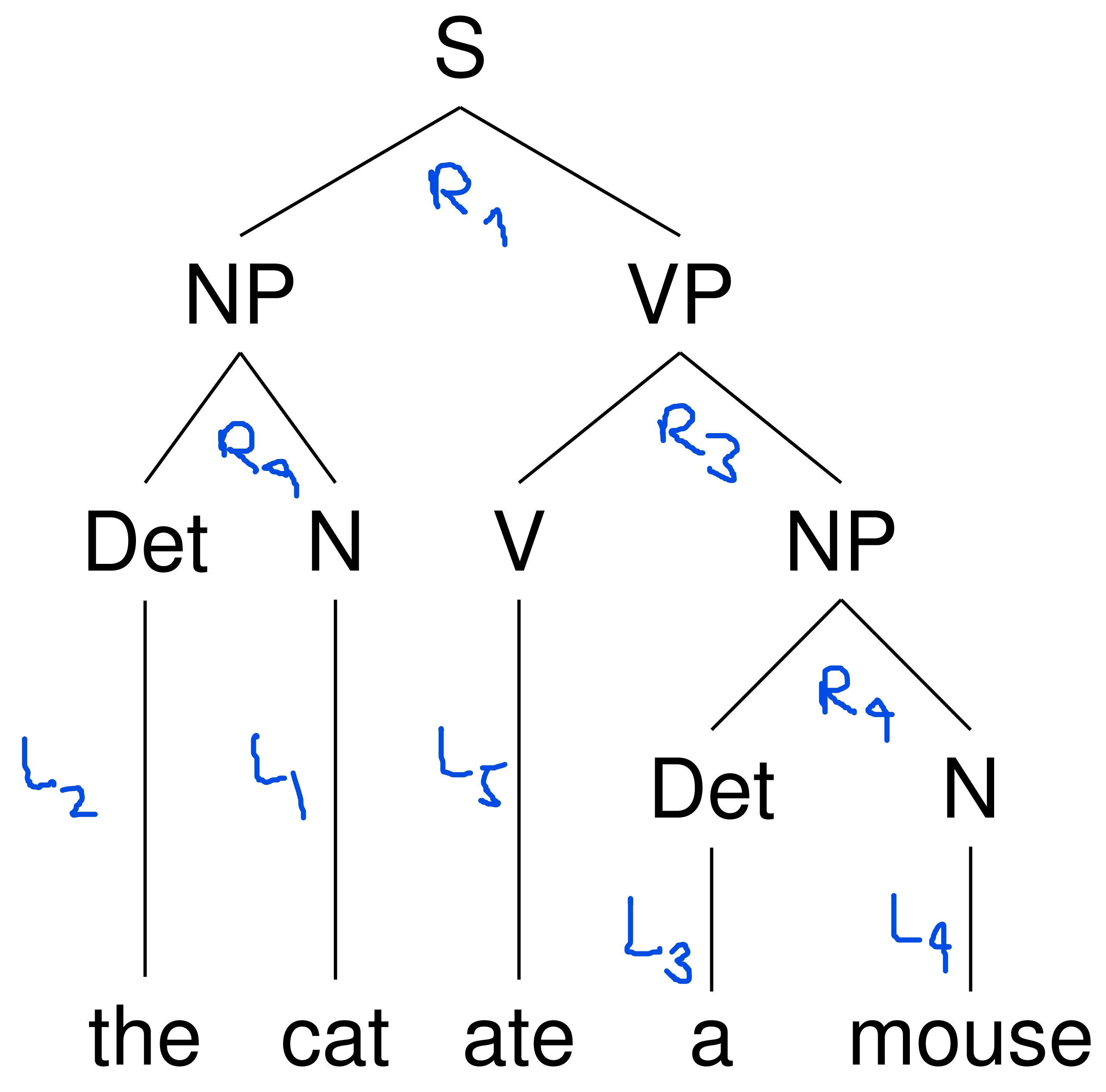


Such a tree will be called a **syntactic tree** (or parse tree, or syntactic structure) associated to W by $\mathcal{G}$.

M. Rajman & J.-C. Chappelier

Syntactic tree(s) associated with a sentence

Example: the derivation (R_1, R_4, L_2, L_1, R_3, L_5, R_4, L_3, L_4) can be represented by the following syntactic tree:



M. Rajman & J.-C. Chappelier
EPFL

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.

M. Rajman & J.-C. Chappelier
EPFL

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:

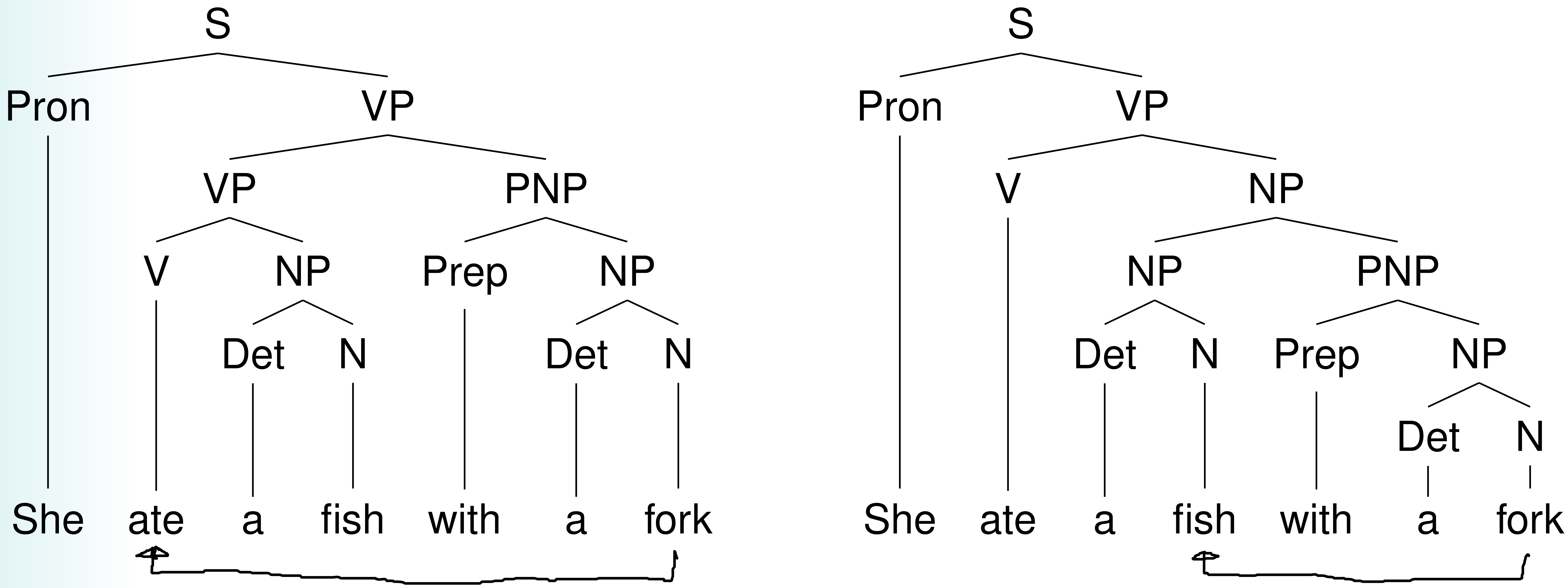
- words are often associated with several parts-of-speech (for example "time" can be a verb or a noun).
  This can lead to multiple syntactic interpretations corresponding to global structural ambiguities
  Example: time flies like an arrow
- word attachments are often not completely constrained at syntactic level. This can lead to multiple syntactic interpretations corresponding to local structural ambiguities
  Example: She ate a fish with a fork

(more)

M. Rajman & J.-C. Chappelier

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 (late 60's)

Input	Output	Resource
sentence	{ trees (analyser) yes/no (recognizer)	CFG

The CYK algorithm

CYK is a bottom-up chart parsing algorithm characterized by 3 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;
- it can provide partial analysis of syntactically correct subsequences of syntactically incorrect sequences.

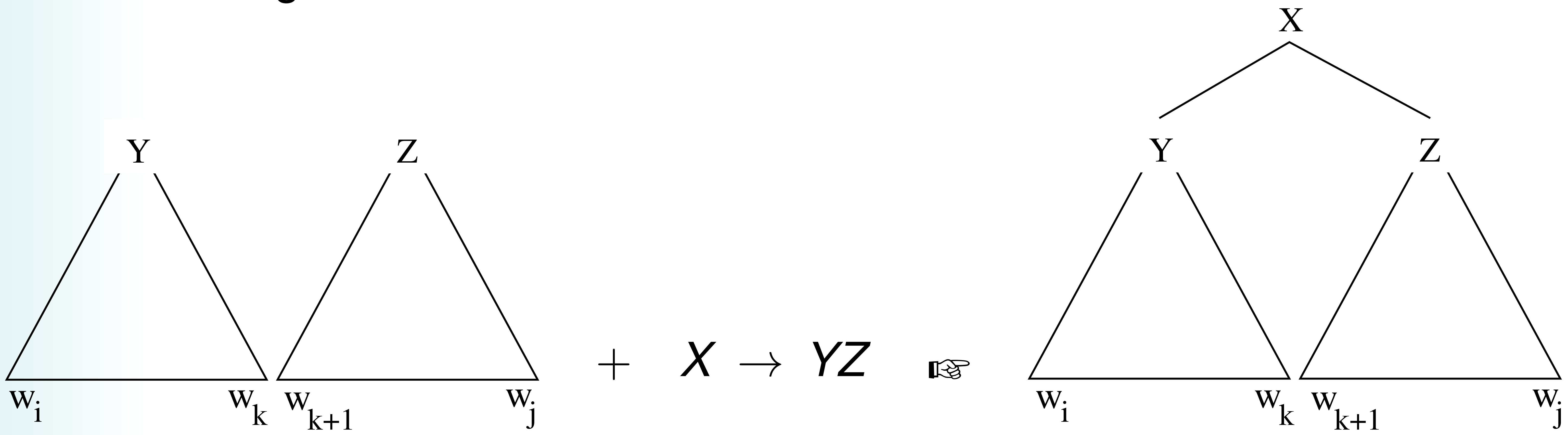
However, its standard implementation suffers from two important drawbacks:

- 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.

Subsequence interpretations are built 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. → Binarization

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 $X \in \mathcal{C} \setminus \mathcal{T}$ and $X_1, X_2 \in \mathcal{C}$

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

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

where $X \in \mathcal{C} \setminus \mathcal{T}$ and $X_1, X_2 \in \mathcal{C}$

M. Rajman & J.-C. Chappelier

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

M. Rajman & J.-C. Chappelier

CYK algorithm: basic principles (2)

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

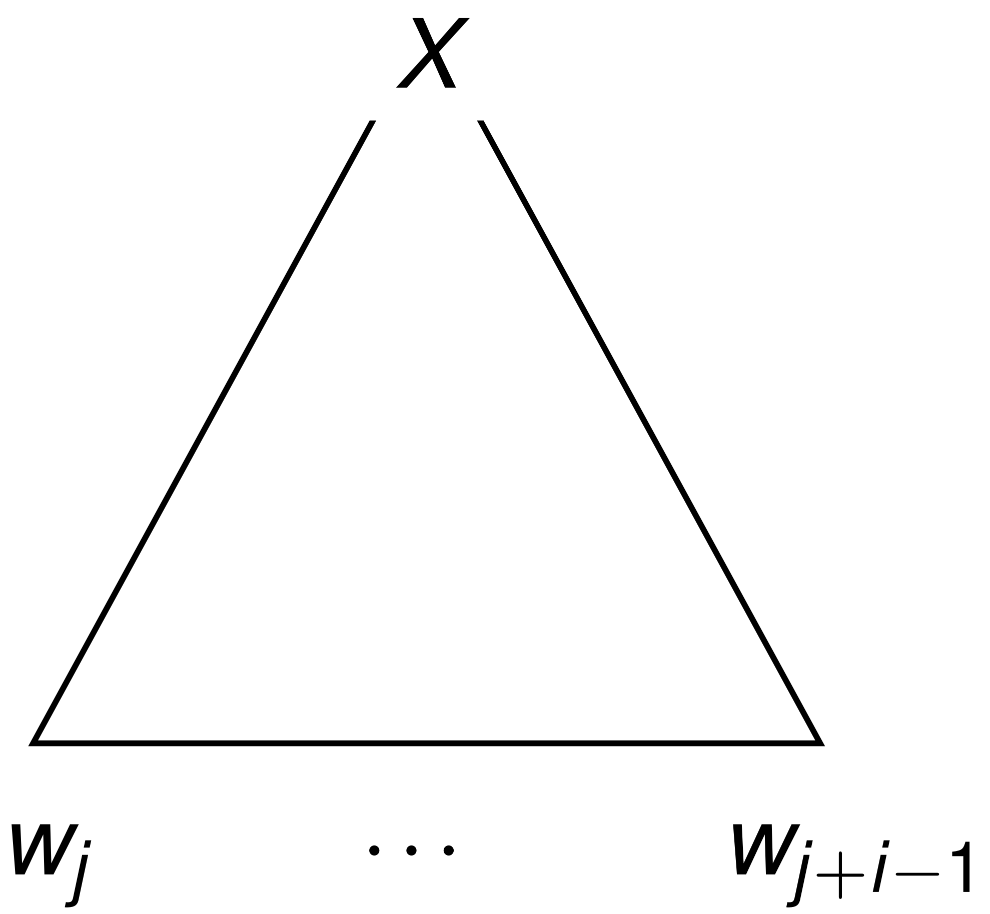
if the grammar is in CNF (or in eCNF) the computation of the syntactic interpretations of a sequence W of length n only requires the exploration of all the decompositions of W into exactly **two** sub-subsequences, each of them corresponding to a cell in a chart. The number of pairs of sub-sequences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

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								
7								
6								
5								
4								
3								
2								
1								
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

CYK Algorithm: principle

L_2: Det $\rightarrow$ the

8								
7								
6								
5								
4								
3								
2								
1	Det							
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

CYK Algorithm: principle

L_1: N → cat

8								
7								
6								
5								
4								
3								
2								
1	Det	N						
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

M. Rajman & J.-C. Chappelier

CYK Algorithm: principle

L_5: V $\rightarrow$ ate

R_5: VP $\rightarrow$ V

8								
7								
6								
5								
4								
3								
2								
1	Det	N	V, VP					
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

CYK Algorithm: principle

8								
7								
6								
5								
4								
3								
2								
1	Det	N	V, VP	Det	N	Prep	Det	N
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

...

M. Rajman & J.-C. Chappelier

CYK Algorithm: principle

R_2: NP $\rightarrow$ Det N

$R_{3.2}$: X_1 $\rightarrow$ Det N

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

M. Rajman & J.-C. Chappelier

EPFL

CYK Algorithm: principle

R_2: NP $\rightarrow$ Det N

$R_{3.2}$: X_1 $\rightarrow$ Det N

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

CYK Algorithm: principle

8								
7								
6								
5								
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
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

R_2: S $\rightarrow$ NP VP

R_6: VP $\rightarrow$ V NP

$R_{7.2}$: $X_2 \rightarrow$ V NP

R_4: PNP $\rightarrow$ Prep NP

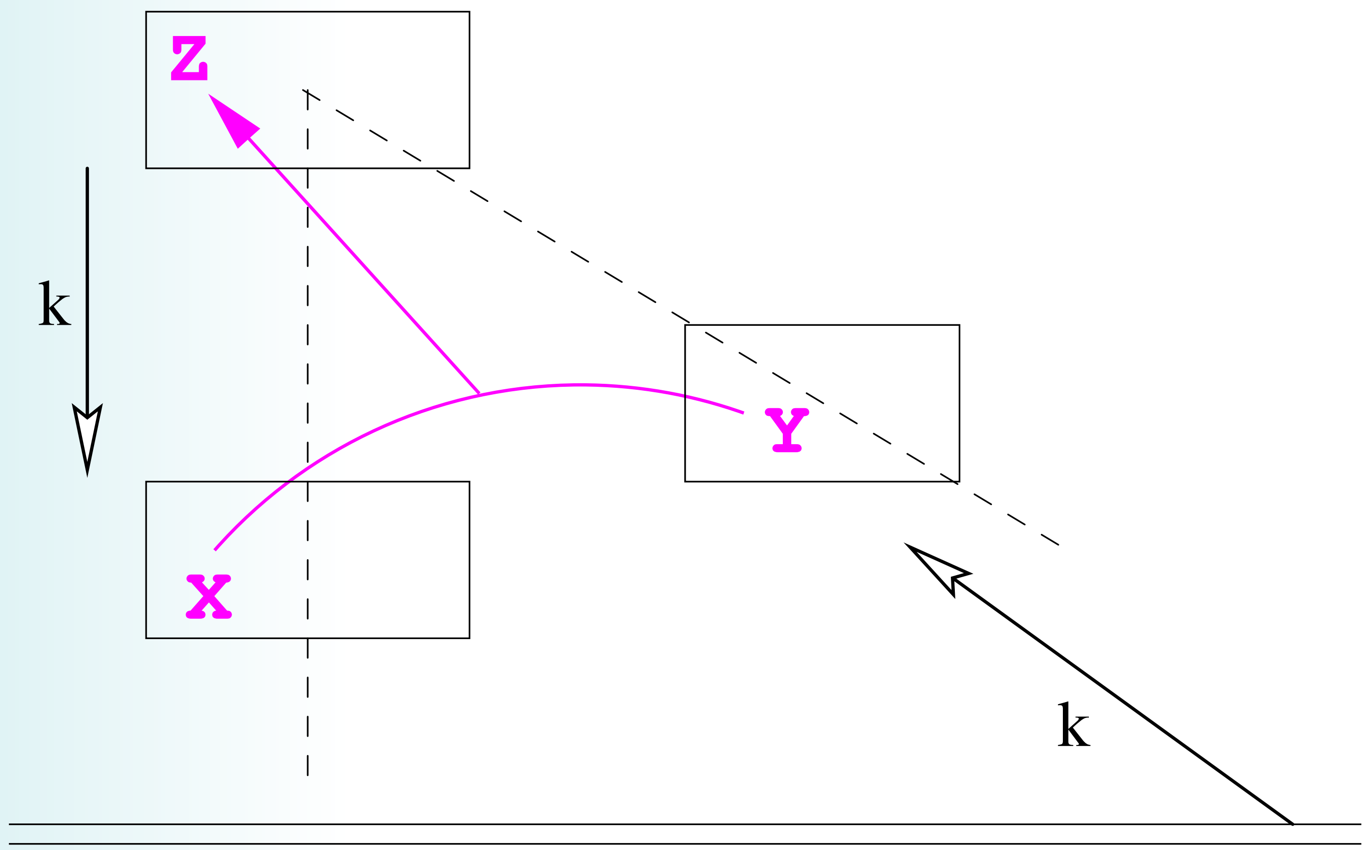
CYK Algorithm: principle

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
i/j	1	2	3	4	5	6	7	8
	the	cat	ate	a	mouse	in	the	garden

Formal algorithm

1) Initialisation: fill first row with corresponding Part-of-Speech

2) Fill chart:



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 $X \in \text{chart}[i-k][j]$ **do**
        for all $Y \in \text{chart}[k][i+j-k]$ **do**
          for all $Z \to X\ Y \in \mathcal{R}$ **do**
            Add Z to chart$[i][j]$

M. Rajman & J.-C. Chappelier

Analyzer or recognizer?

- The preceding algorithm does not store the parse trees.
  ➥ Recognizer (check whether 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

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

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:

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

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

There are modified versions of the CYK algorithm where CNF is no longer required (☞ $\mathcal{C}$ is then smaller): bottom-up chart parsing

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

M. Rajman & J.-C. Chappelier

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

Beyond CNF: bottom-up chart parting

Idea: get rid of (e)CNF constraint

How to?

☞ on-line binarization, when needed, during bottom-up analysis

Mainly:

- factorize (with respect to α) all the partial derivations $X \rightarrow \alpha \bullet \beta$ ☞ $\alpha \bullet \ldots$
  This is possible because processing bottom-up.

[α and β are (non-empty) sequences of non-terminals.]

Bottom-up Chart Parsing

More formally, a CYK algorithm in which:

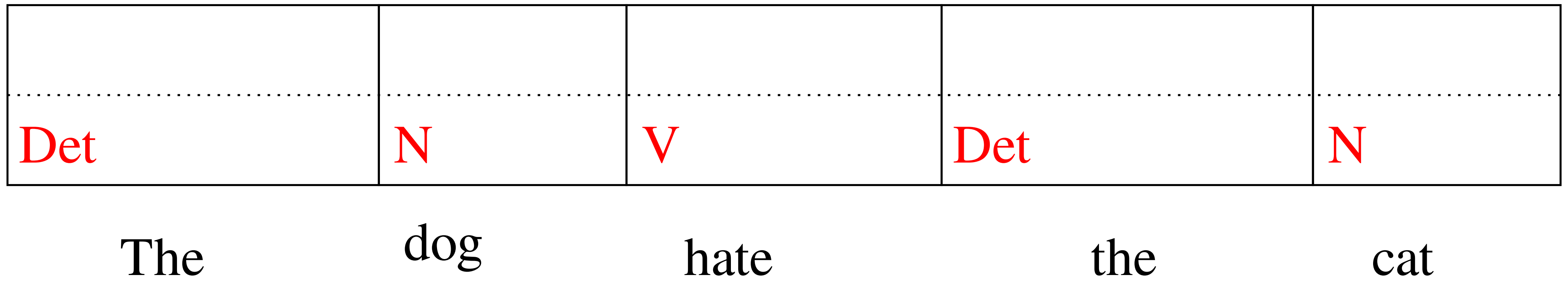
- cells contain **two** kind of objects:
  $[\alpha \bullet ..., i, j]$ and $[X, i, j]$ respectively
- initialization consists in adding $[X, i, j]$ for all $X \rightarrow w_{ij} \in \mathcal{R}$
  (w_{ij} is a sequence of tokens of the input sentence;
  see "*Dealing with compounds*" later slide)
- and the completion phase becomes:
  (association of two cells)

$$[\alpha \bullet ..., i, j] \oplus [X, k, j+i] \Rightarrow \begin{cases} [\alpha X \bullet ..., i+k, j] & \text{if } Y \rightarrow \alpha X \beta \in \mathcal{R} \\ [Y, i+k, j] & \text{if } Y \rightarrow \alpha X \in \mathcal{R} \end{cases}$$

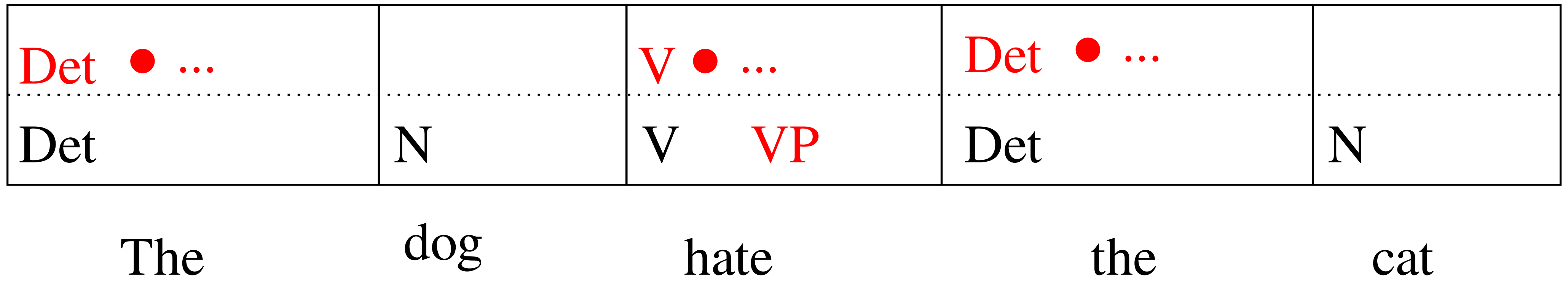
("self-filling")

$$[X, i, j] \Rightarrow \begin{cases} [X \bullet ..., i, j] & \text{if } Y \rightarrow X \beta \in \mathcal{R} \\ [Y, i, j] & \text{if } Y \rightarrow X \in \mathcal{R} \end{cases}$$

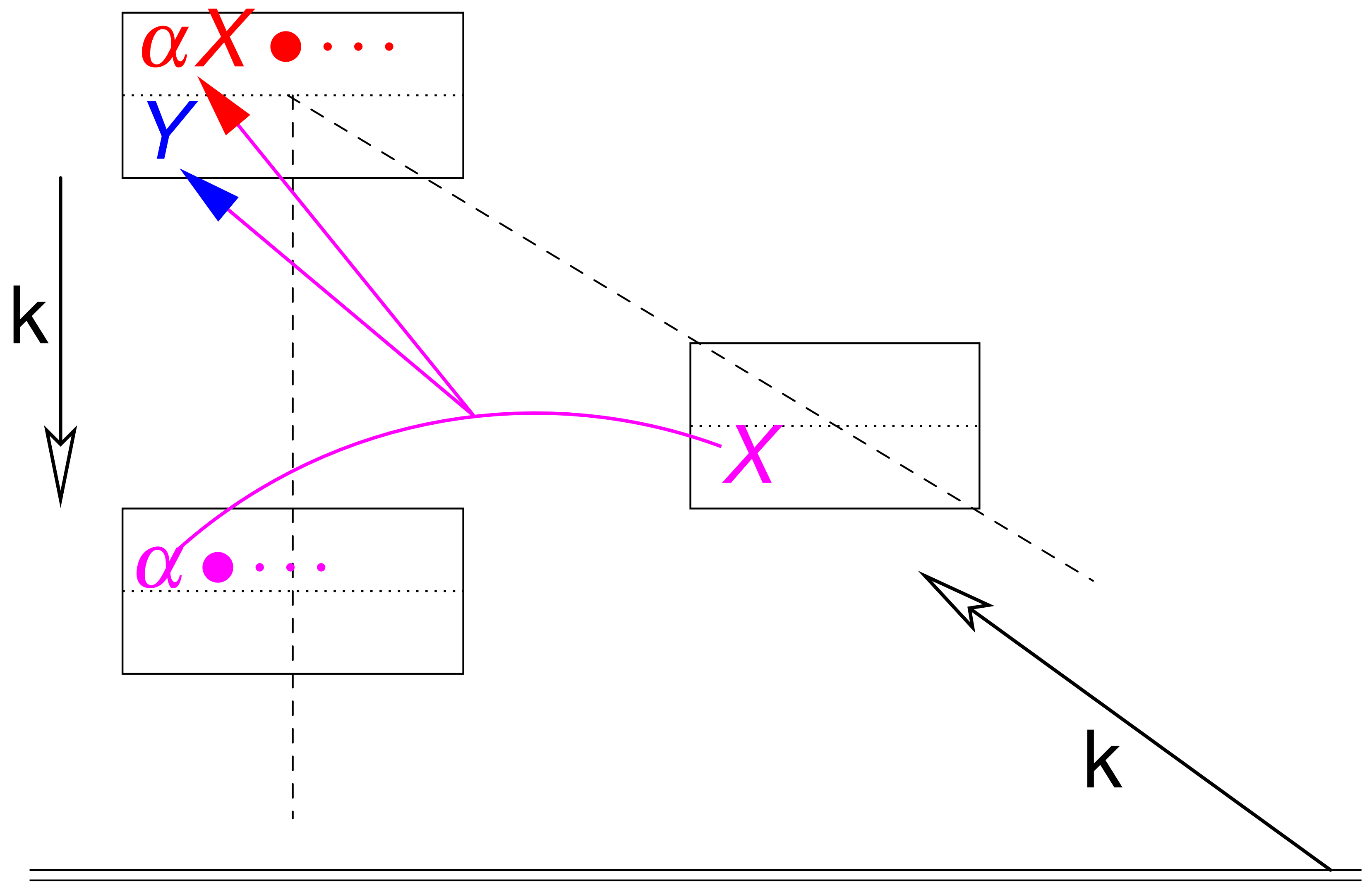
Bottom-up Chart Parsing: illustration



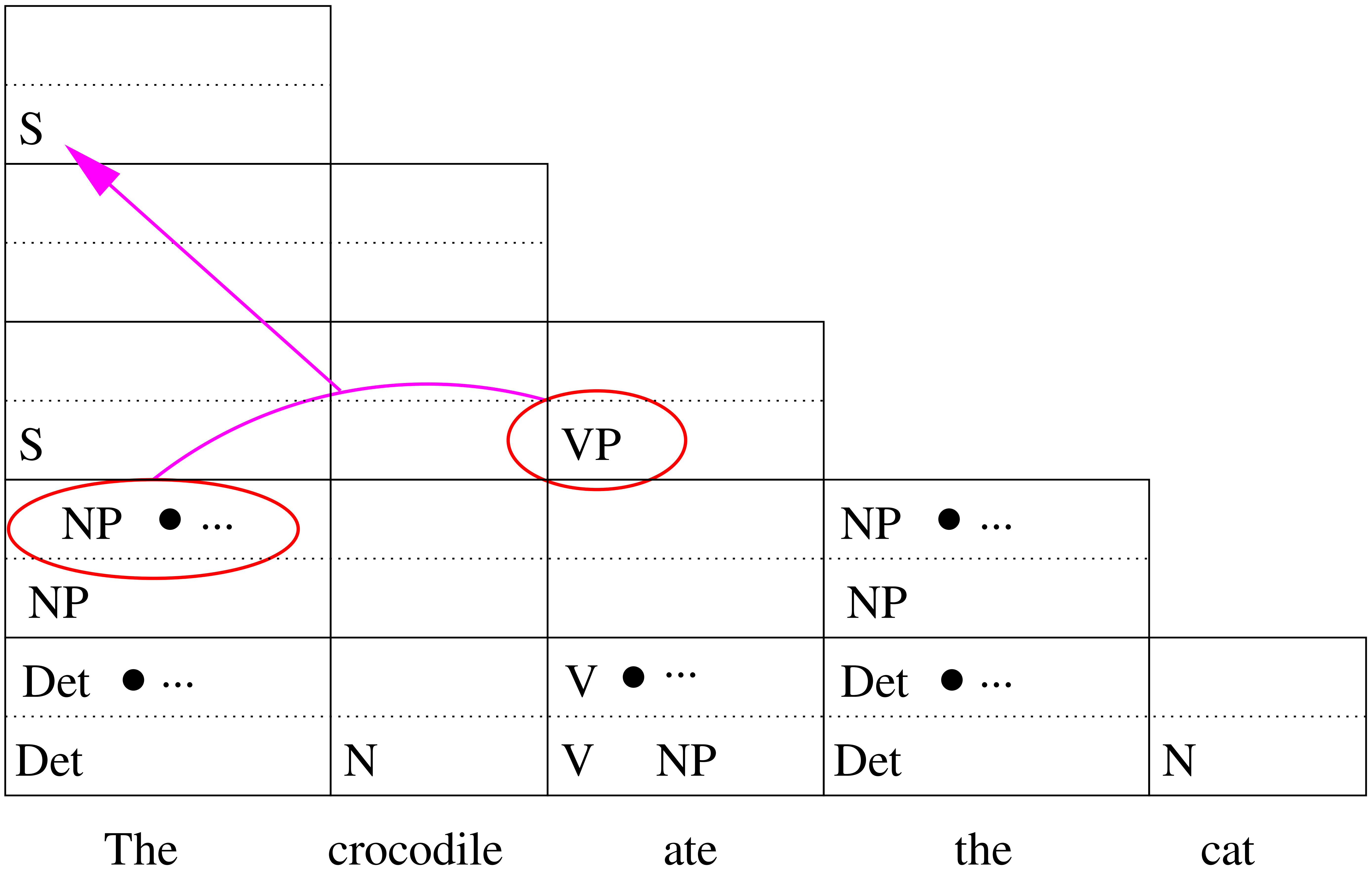
Initialization:



Completion:

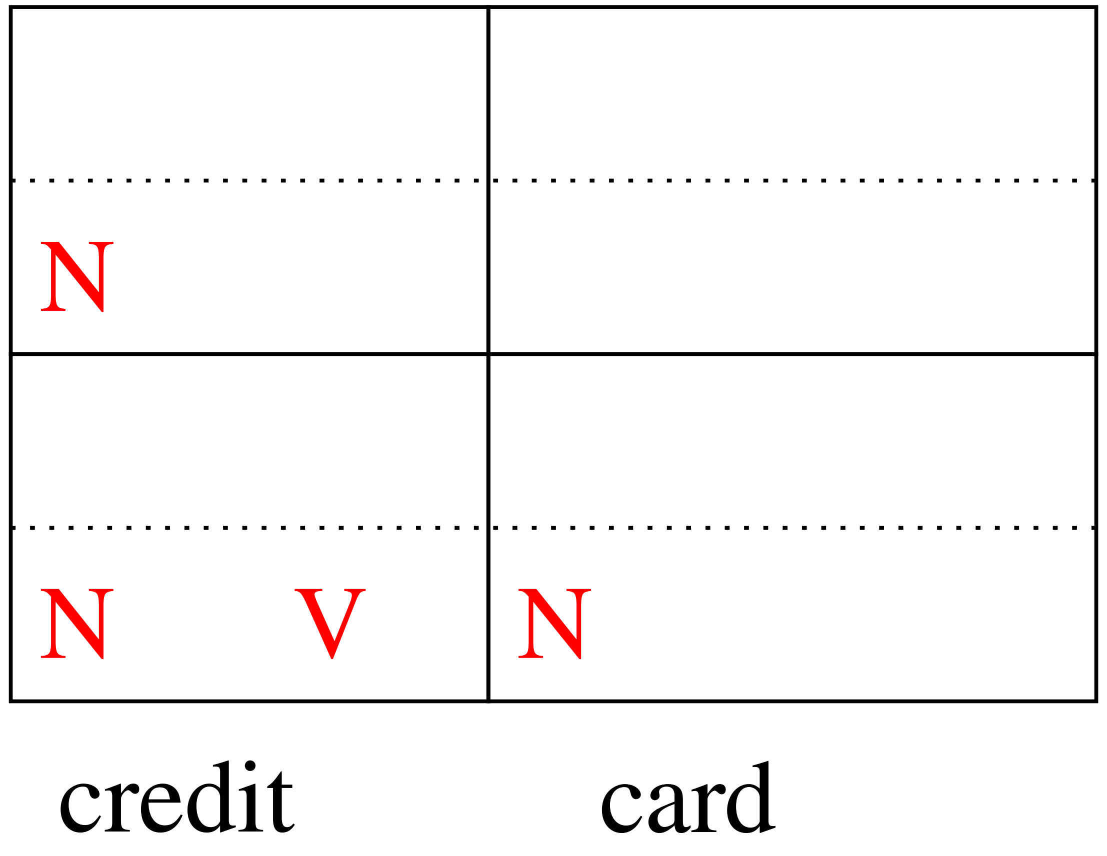


Bottom-up Chart Parsing: Example



Dealing with compounds

Example on how to deal with compouds during initialization phase:



M. Rajman & J.-C. Chappelier

Keypoints

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

M. Rajman & J.-C. Chappelier

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)