Computational Linguistics 2014-2015

- **Walter Daelemans** (walter.daelemans@uantwerpen.be)
- **Guy De Pauw** (guy.depauw@uantwerpen.be)
- **Mike Kestemont** (mike.kestemont@uantwerpen.be)

http://www.clips.uantwerpen.be/cl1415

Practical

Location	P0.11 (Scribanihuis)
Reading material	• D. Jurafsky & J.H. Martin (2009) Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (2nd ed). Pearson Education, USA. • Natural Language Processing with Python
Software	Python 3.4 and NLTK: Installation Instructions
Evaluation	Take-home assignments and oral examination
Lecturers	Walter Daelemans: walter.daelemans@uantwerpen.be Mike Kestemont: mike.kestemont@uantwerpen.be Guy De Pauw: guy.depauw@uantwerpen.be

Program

Session	Day	Date	Chapter	Topic	Reading Assignment	Slides	Take-home Assignment
1	Monday	29/9/2014	**Python**	Session 1 - Variables	See Github		
2	Thursday	2/10/2014	**Python**	Session 2 - Collections			
3	Monday	6/10/2014	**Python**	Session 3 - Conditions (and an introduction to loops)			
4	Thursday	9/10/2014	**Python**	Session 4 - Loops			
5	Monday	13/10/2014	**Python**	Session 5 - Reading and writing to files			
6	Thursday	16/10/2014	**Python**	Session 6 - Writing your own Functions and importing packages			
7	Monday	20/10/2014	**Python**	Session 7 - Regular Expressions in Python			
8	Thursday	23/10/2014	**Python**	Session 8 - Advanced looping in Python and list comprehensions			
9	Monday	27/10/2014	**Theory**	Introduction to Computational Linguistics	Jurafsky & Martin: Chapter 1	PDF	
10	Monday	3/11/2014	**Theory**	Regular Expressions and Finite State Automata & Transducers	Jurafsky & Martin: Chapter 2; Chapter 3	Slides morfsegment.py	See last slide. Deadline: 24/11 participles.py
	Monday	10/11/2014	**Remembrance day: no session**				
11	Monday	17/11/2014	**Theory**	Part-of-Speech Tagging	Jurafsky & Martin: Chapter 5 (not 5.5, 5.8 and 5.9)	Slides Python Code	See last slide. Deadline: 8/12
12	Monday	24/11/2014	**Theory**	Syntactic Analysis & Parsing	Jurafsky & Martin: Chapter 12 (not 12.7.2, 12.8); Chapter 13 (not 13.4.1, 13.4.2, 13.5.1)		
13	Monday	1/12/2014	**Theory**	Minimum Edit Distance + Probabilistic Methods	Jurafsky & Martin: Chapter 3.11; Chapter 4.1, 4.2 and 4.3; Chapter 5.5 and 5.9; Chapter 14.1, 14.3 and 14.4;		
14	Monday	8/12/2014	**Theory**	Word Sense Disambiguation	Jurafsky & Martin: Chapter 19.1, 19.2, 19.3, Chapter 20 (20.1->20.5)		
15	Monday	15/12/2014	**Theory**	Sentence semantics and discourse; Information extraction	Jurafsky & Martin: Chapter 21; Chapter 22		

Syntactic Analysis
Parsing

Parser vs Grammar

Grammar

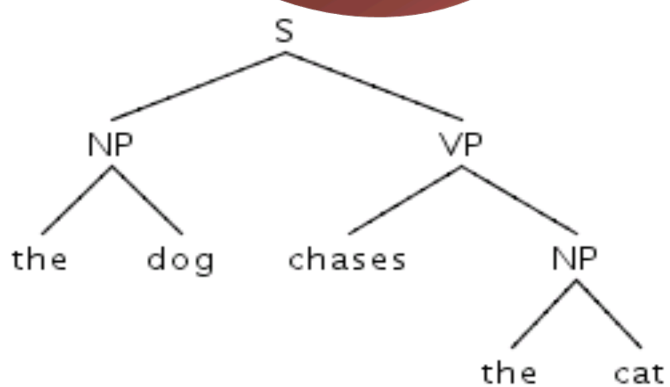
CFG,

dependency,

…

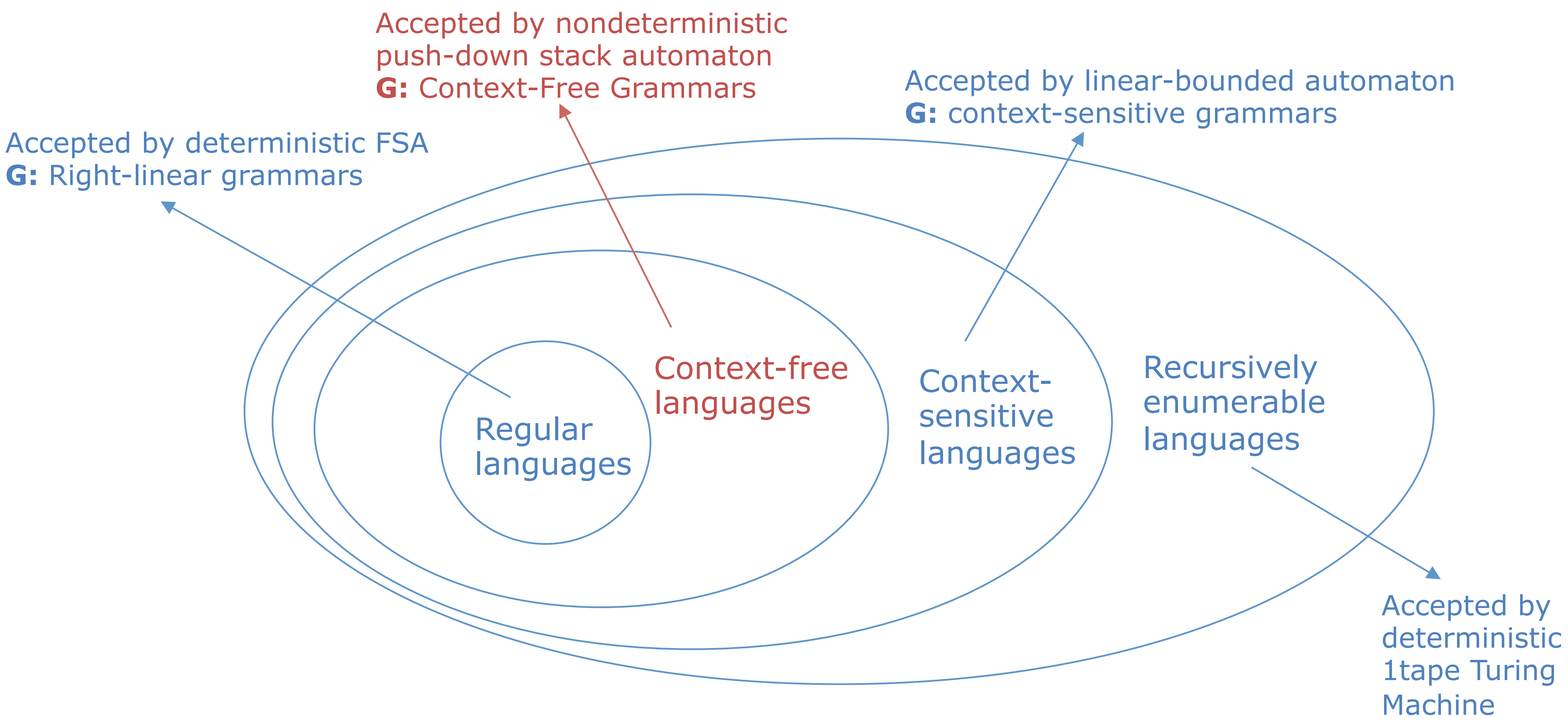
Parser

top-down vs bottom-up, depth-first vs breadth-first, chart-parsing, Tomita



Syntactic Analysis

The Chomsky Hierarchy



Context-Free Grammar

N set of non-terminal symbols **[constituents]**

Σ set of terminal symbols **[pos, words]**

R set of rules / productions **[rewrite rules]**

$A \rightarrow \beta$ with $A \in N$

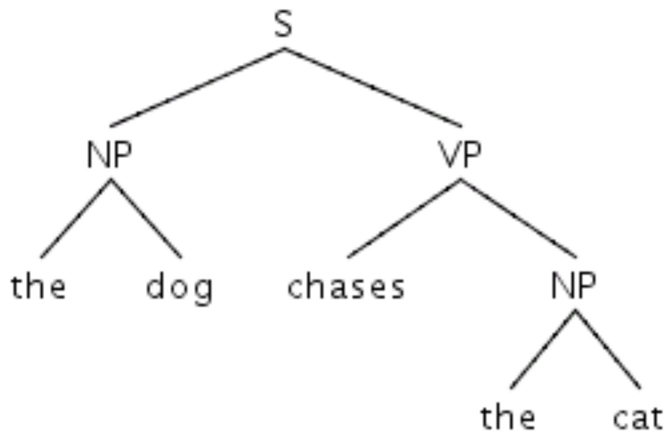
with $\beta \in N \cup \Sigma$

S designated start symbol

Context-Free Grammar (CFG)

- Re-write rules:
  1. rewrite left-hand symbol as right-hand (top-down)
  2. rewrite right-hand context as left-hand-symbol (bottom-up)

S → NP VP
NP → the dog | the cat
VP → chases NP



Advantages of CFGs

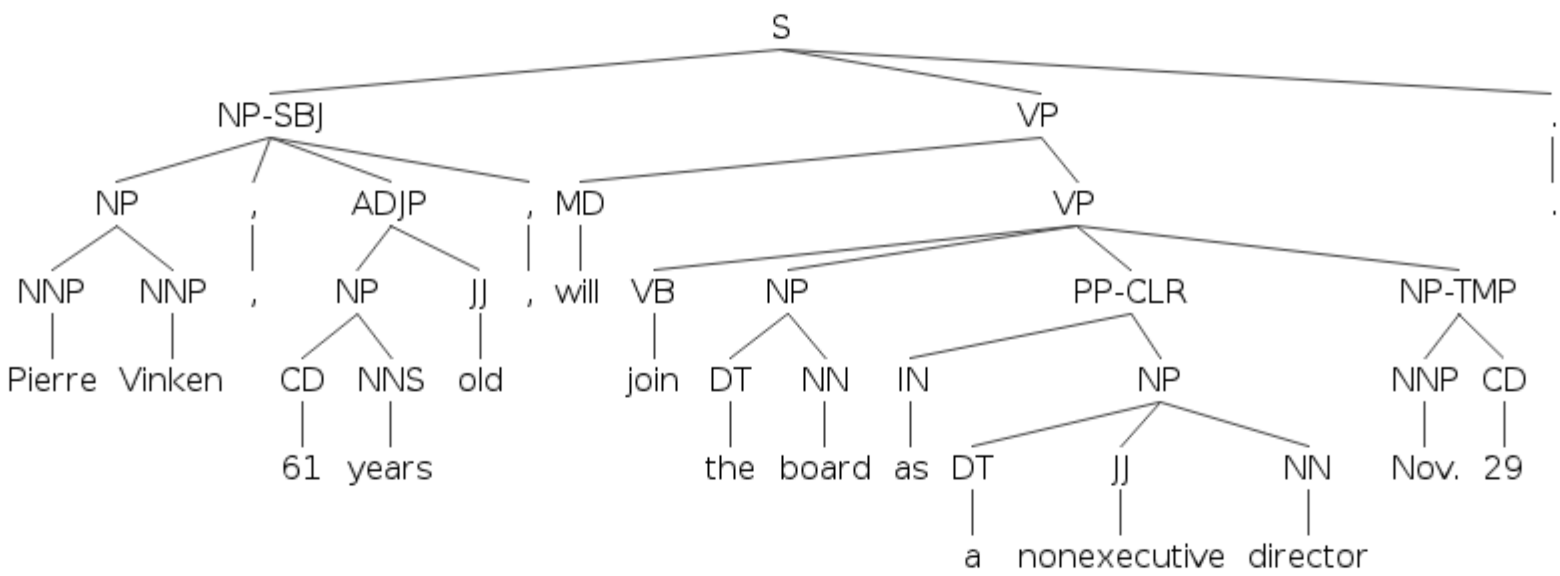
- Powerful enough to describe many structural properties in language

- Well-studied type of formal language
- Formalism is easily applicable in existing parsing algorithms

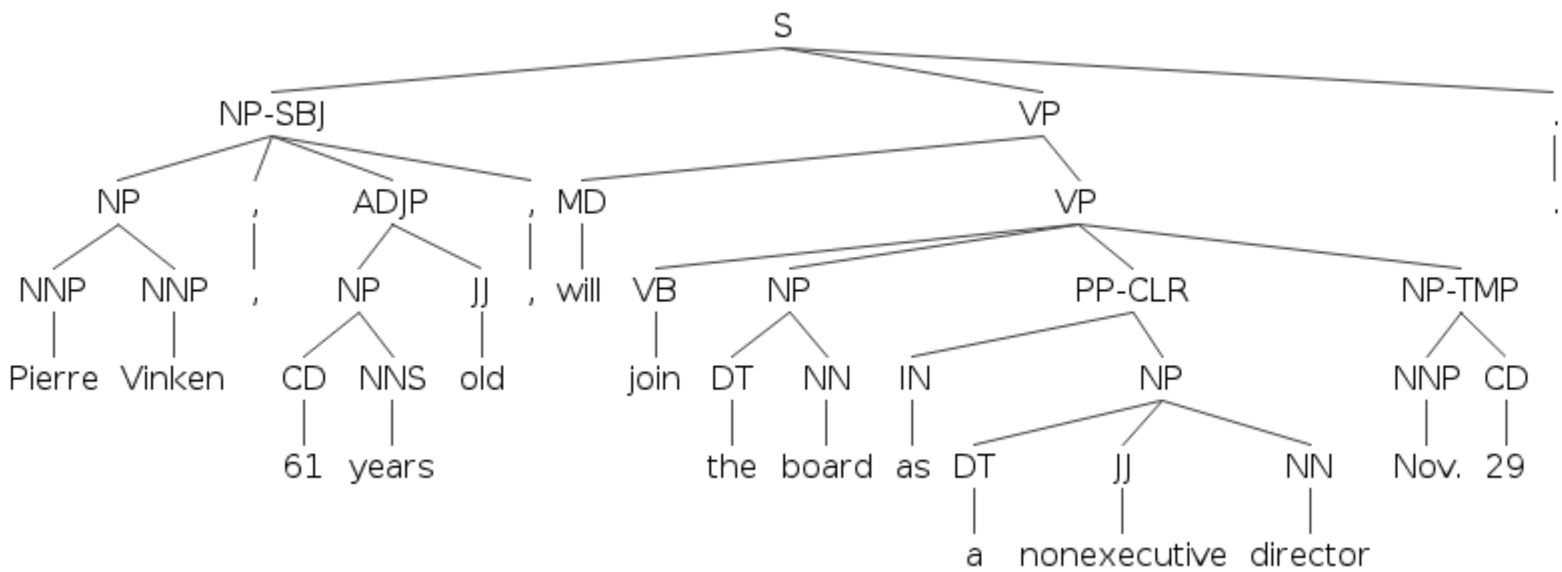
  $\Rightarrow$ differents **parsers** for different **grammars**

Treebanks

- cf. Annotated corpora for data-driven part-of-speech tagging
- Treebank: corpus of syntactically annotated sentences, i.e. collection of tree structures
- e.g. Penn Treebank



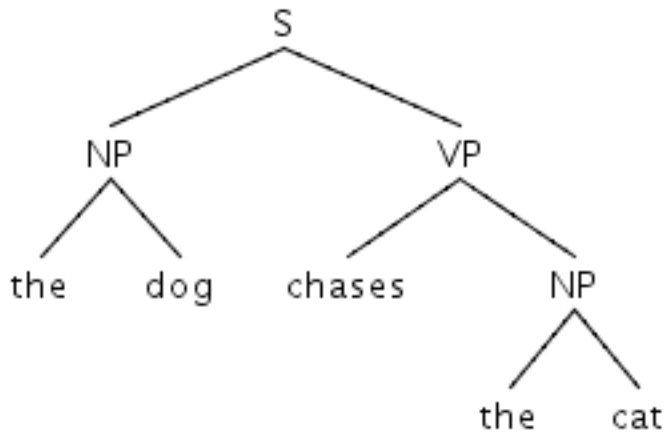
Treebanks




```
(S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) )
```

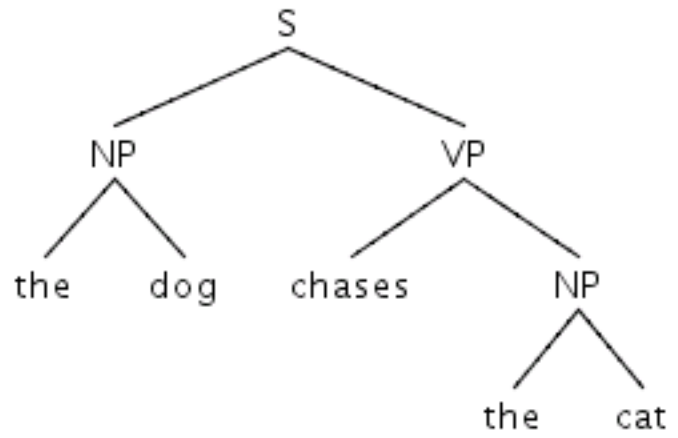
Computational Representation

Tree-Structure



Computational Representation

Tree-Structure

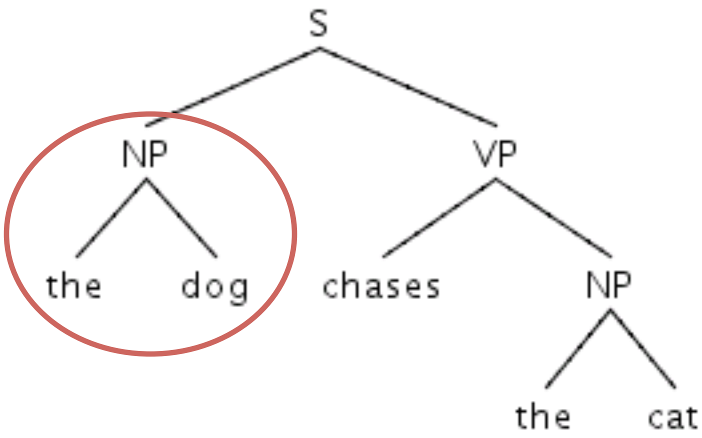


Bracketed

Universiteit Antwerpen

Computational Representation

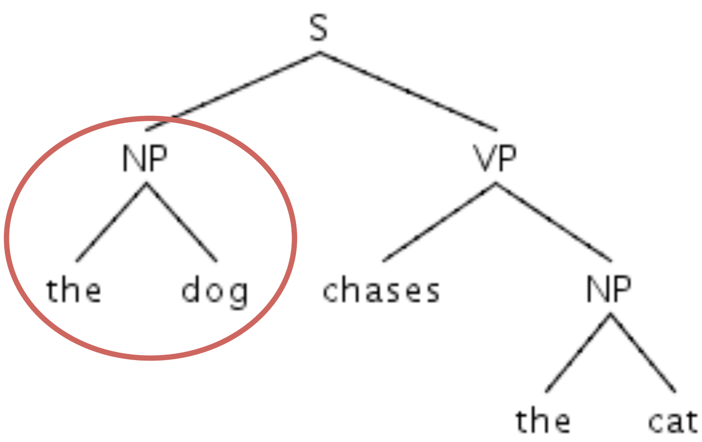
Tree-Structure



Bracketed

Computational Representation

Tree-Structure

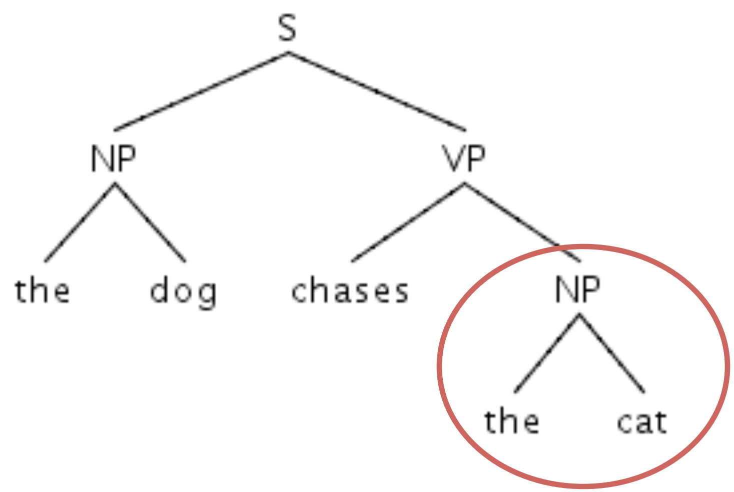


Bracketed

(NP the dog)

Computational Representation

Tree-Structure

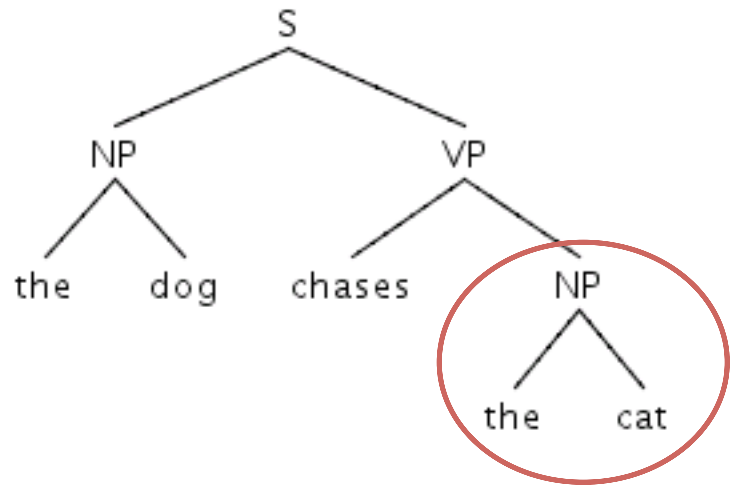


Bracketed

(NP the dog)

Computational Representation

Tree-Structure



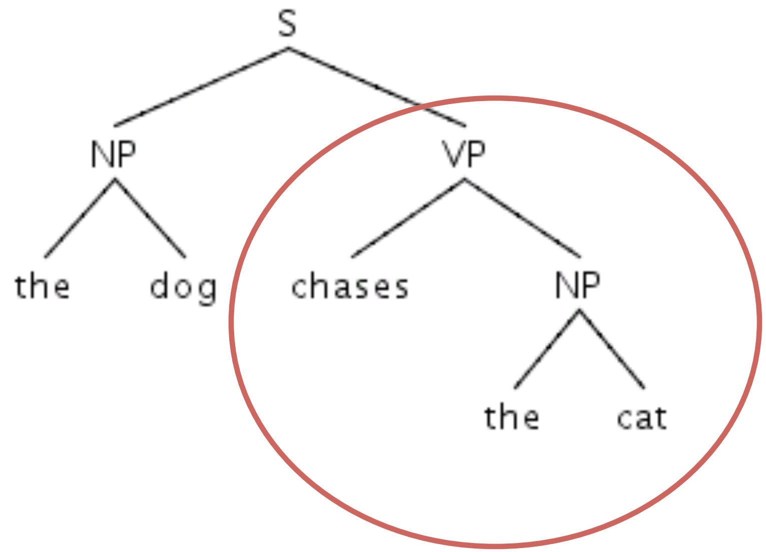
Bracketed

(NP the dog)

(NP the cat)

Computational Representation

Tree-Structure



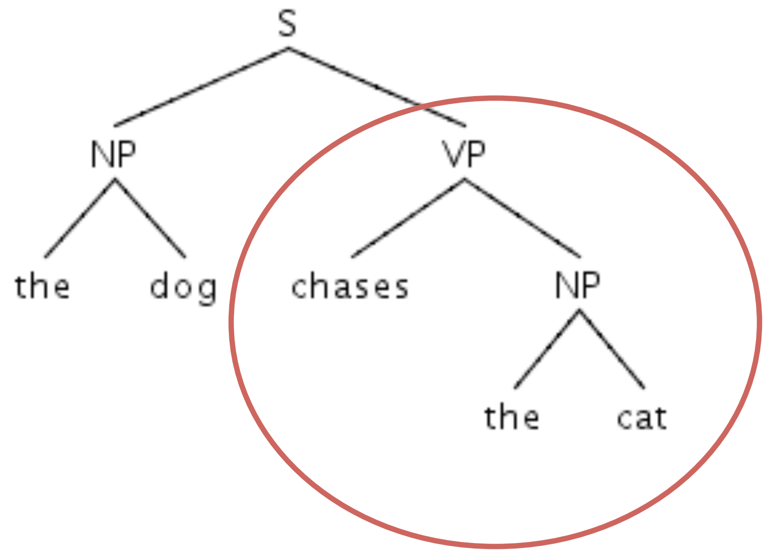
Bracketed

(NP the dog)

(NP the cat)

Computational Representation

Tree-Structure



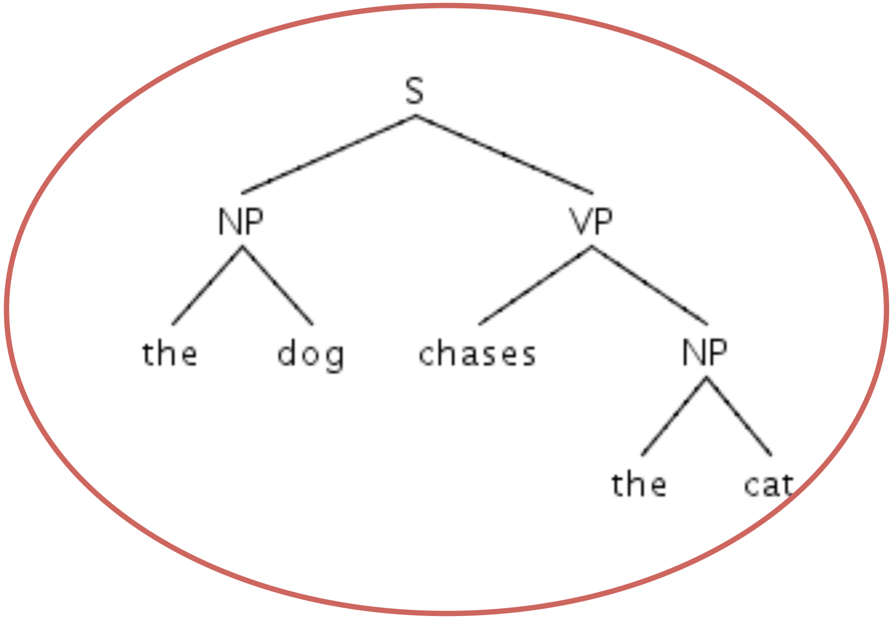
Bracketed

(NP the dog)
(VP chases

(NP the cat))

Computational Representation

Tree-Structure



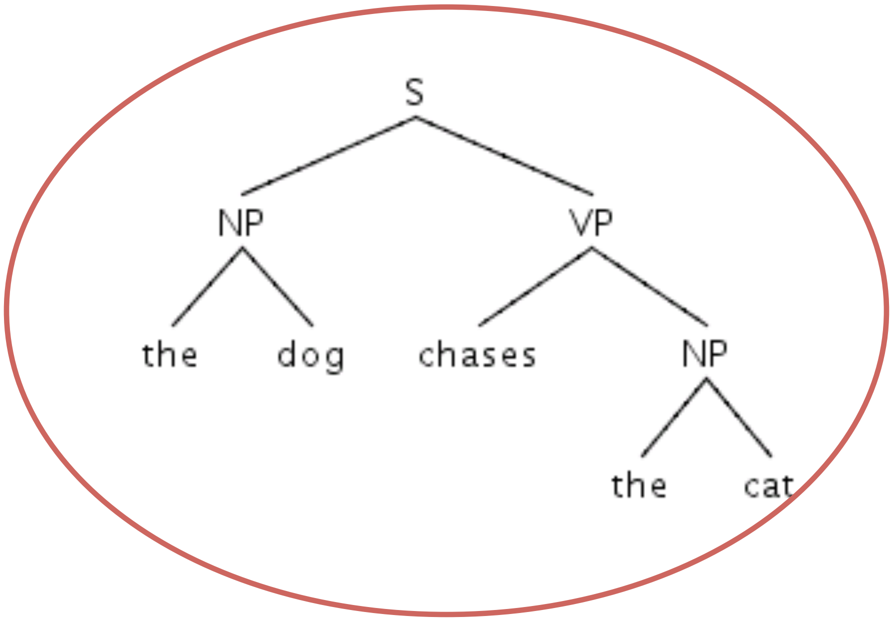
Bracketed

(NP the dog)
(VP chases

(NP the cat))

Computational Representation

Tree-Structure



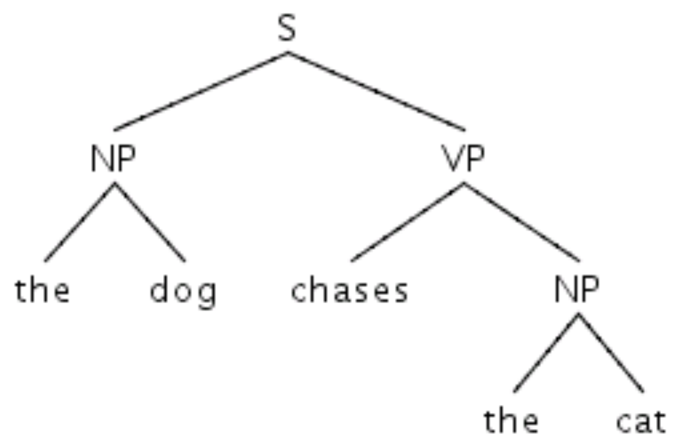
Bracketed

(S (NP the dog)
(VP chases

(NP the cat)))

Computational Representation

Tree-Structure



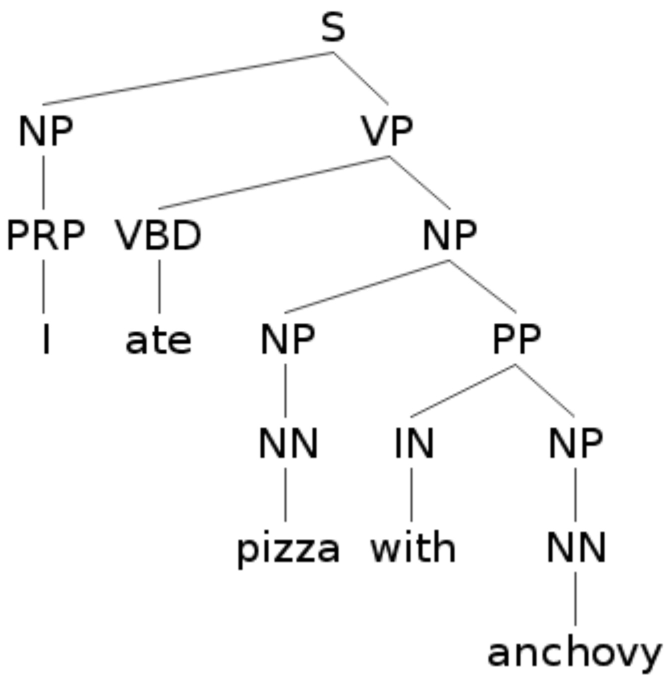
Bracketed

(S (NP the dog)
(VP chases

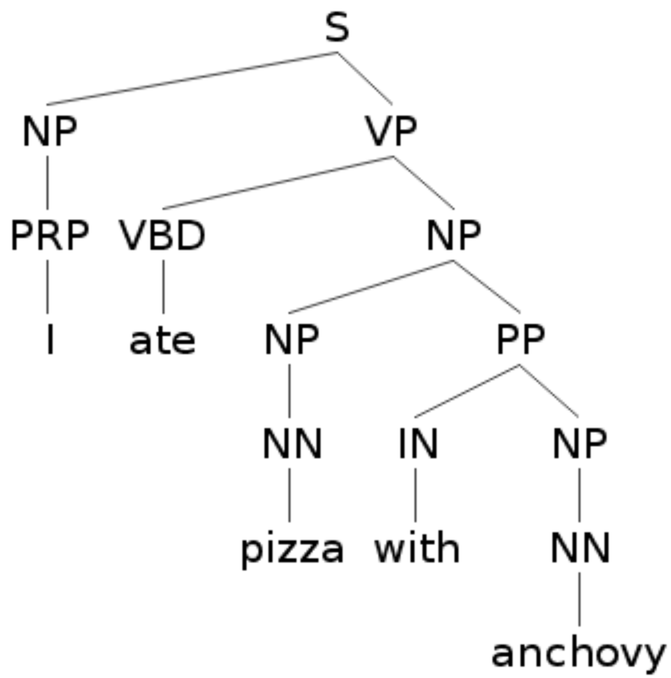
(NP the cat)))

or: (S (NP the dog) (VP chases (NP the cat)))

Exercise: transform the following tree structure into a bracketed representation



Exercise: transform the following tree structure into a bracketed representation



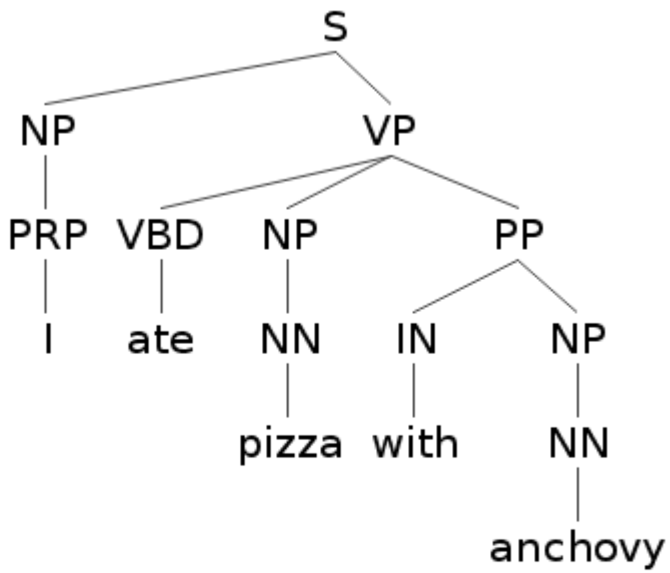
(S (NP (PRP I)) (VP (VBD ate) (NP (NP (NN pizza)) (PP (IN with) (NP (NN anchovy))))))

Exercise: transform the following bracketed structure into a tree-structure

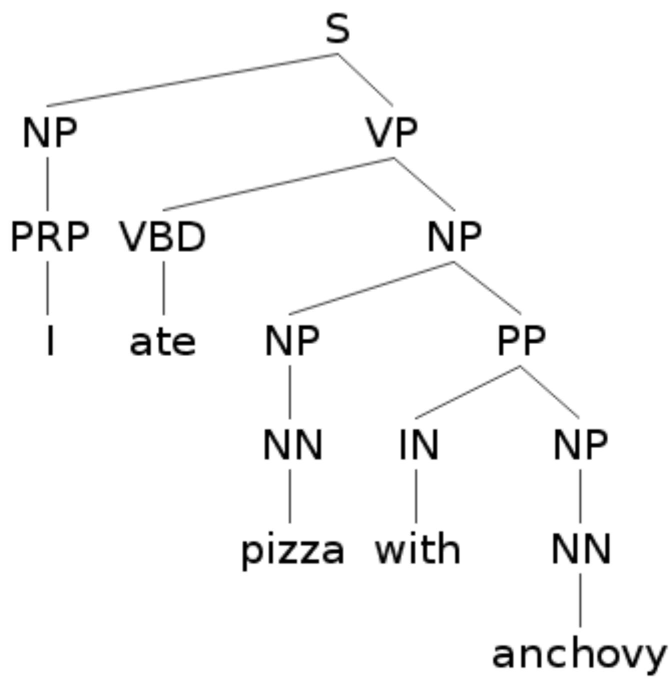
(S (NP (PRP I)) (VP (VBD ate) (NP (NN pizza)) (PP (IN with) (NP (NN anchovy)))))

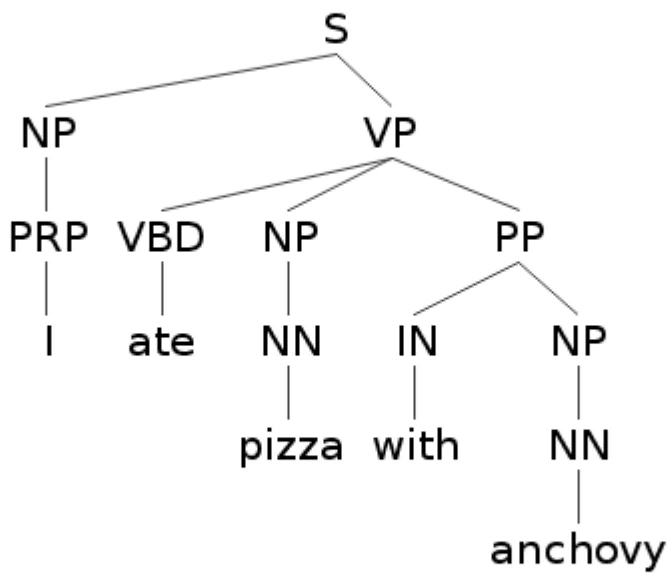
Exercise: transform the following bracketed structure into a tree-structure

(S (NP (PRP I)) (VP (VBD ate) (NP (NN pizza)) (PP (IN with) (NP (NN anchovy)))))



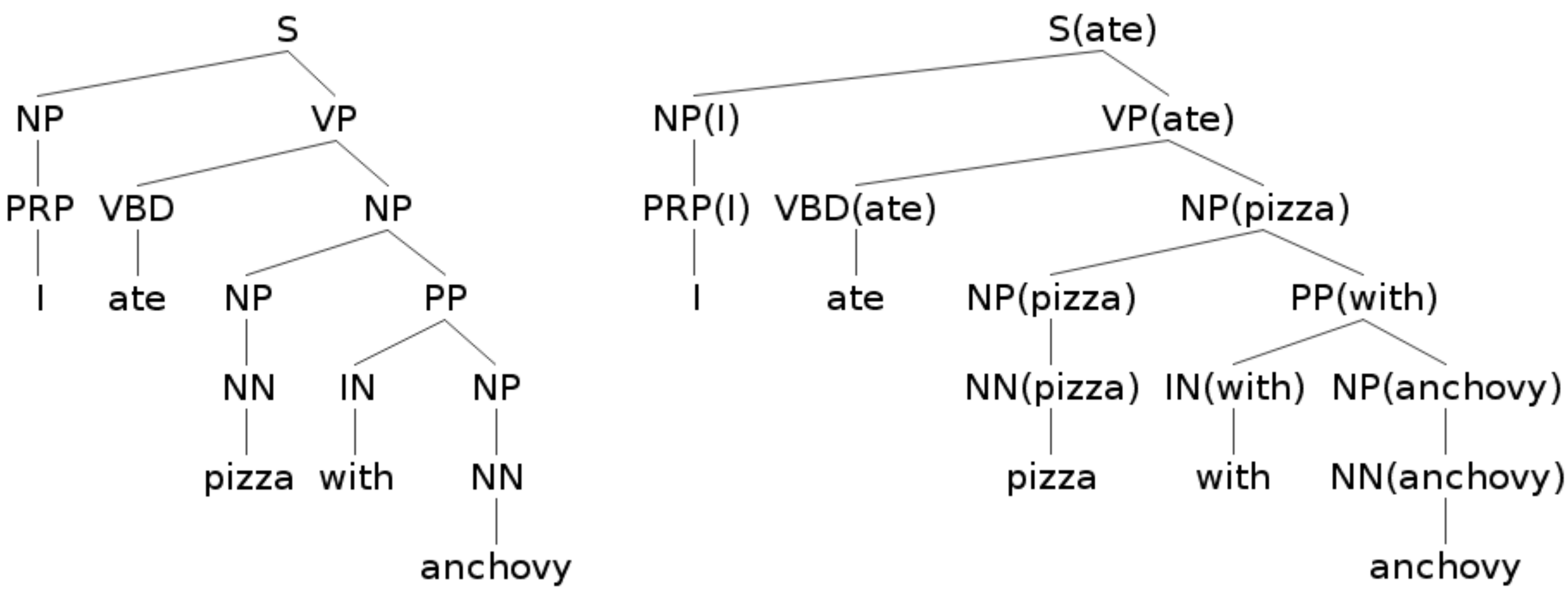
Exercise: what is the difference between the structure in Exercise 1 and the structure in Exercise 2?



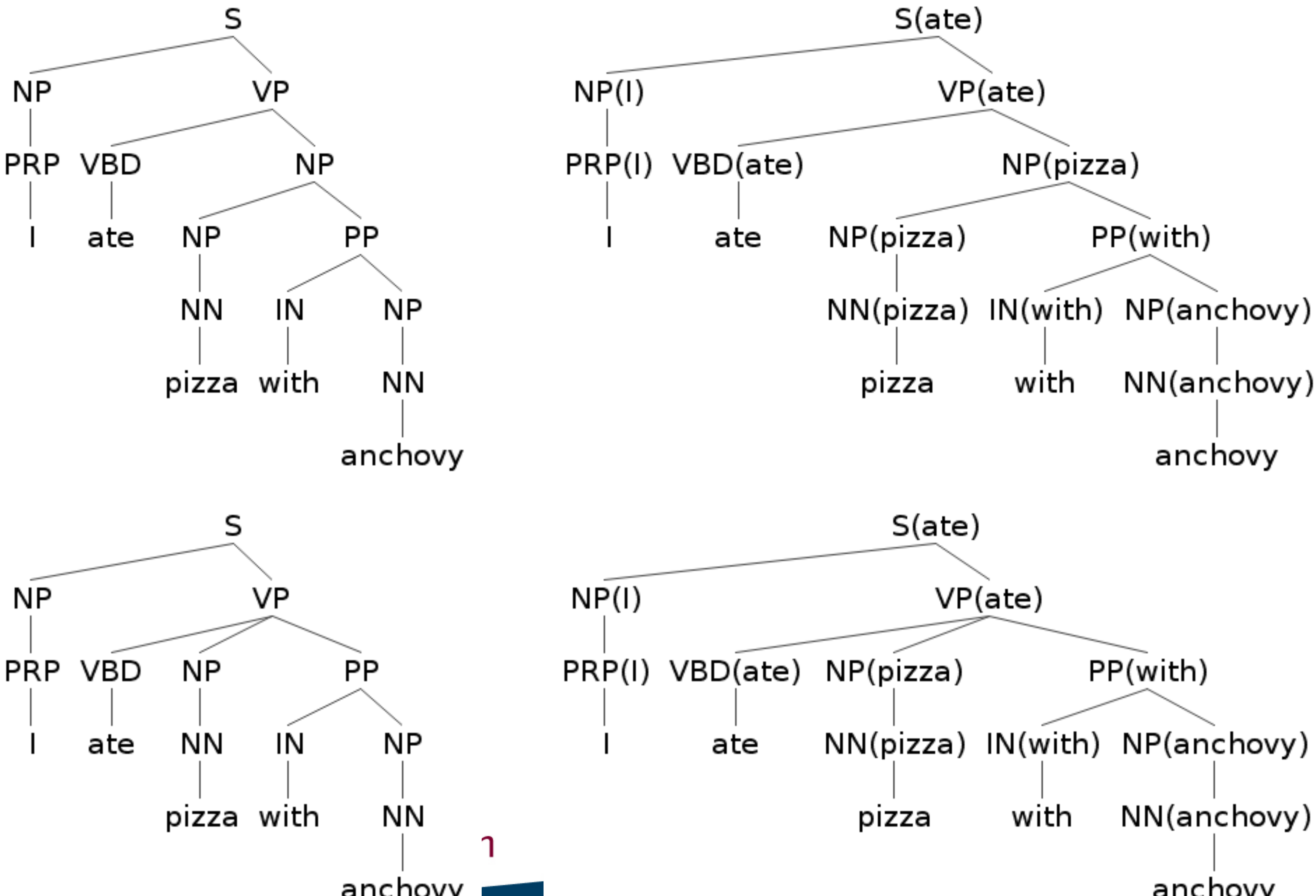


Lexicalized grammar

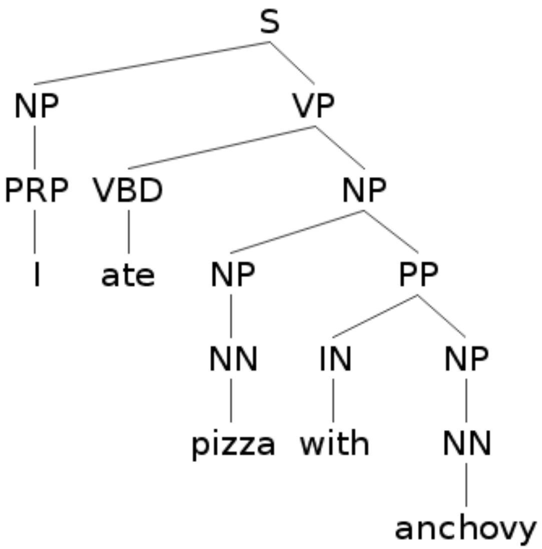
- Identify for each non-terminal category, the head word/part-of-speech tag
- Use this information to *enrich* the tree structure
- VP(VBD,MD,…), NP(NN,NNP,PRP,…), ADJP(JJ), PP(IN)

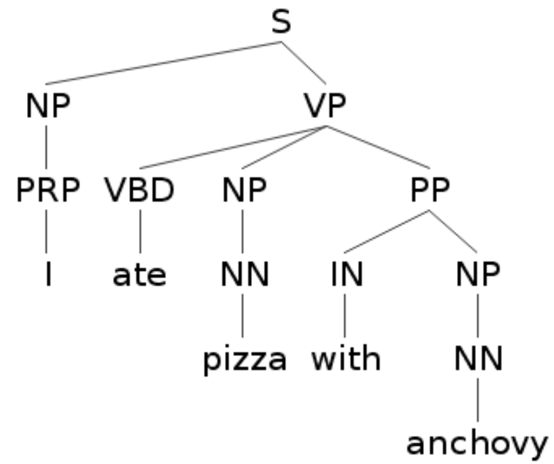


Lexicalized grammar



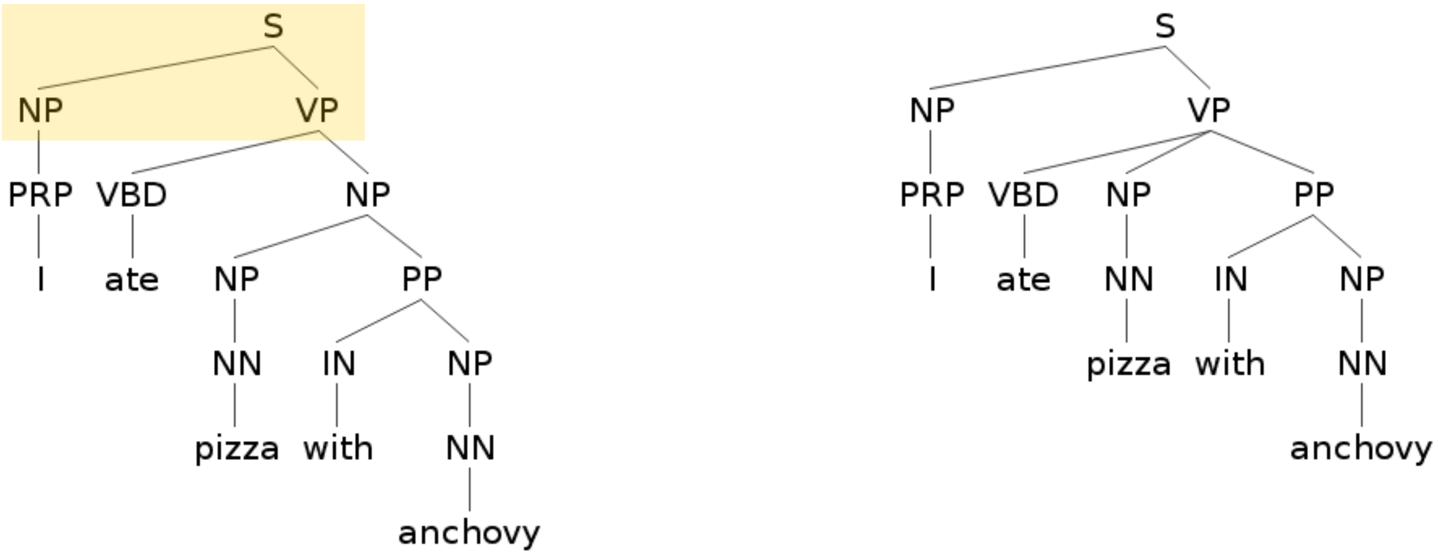
From Treebank to (P)CFG





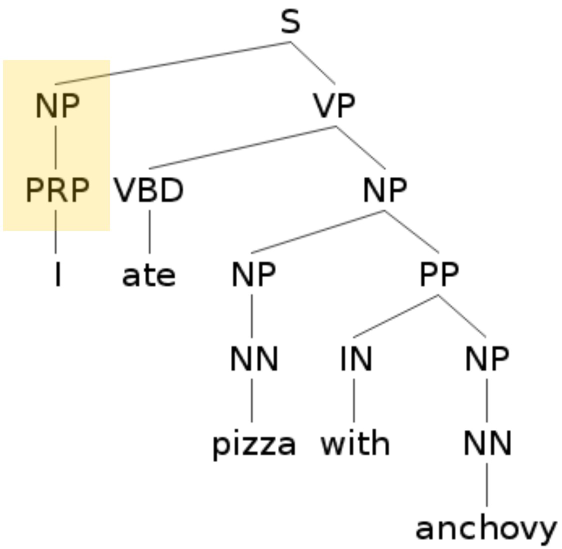
- Cut tree in slices of depth 1
- Do this for each non-terminal in all of the trees of your treebank

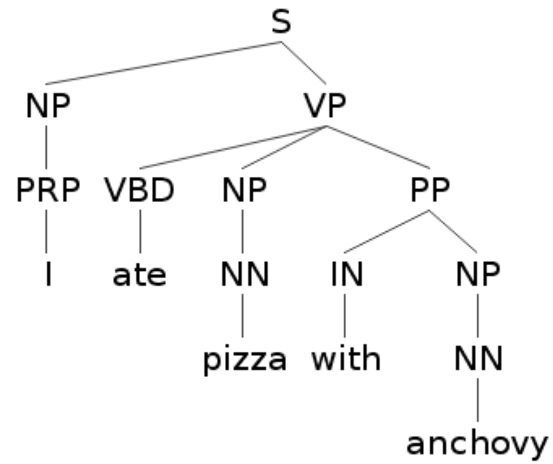
From Treebank to (P)CFG



S → NP VP (1)

From Treebank to (P)CFG

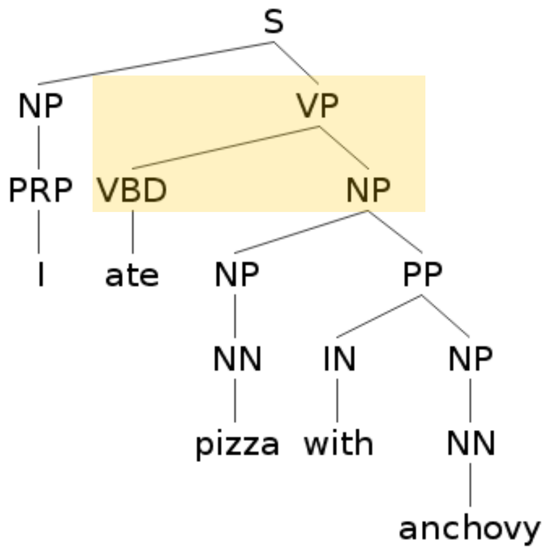


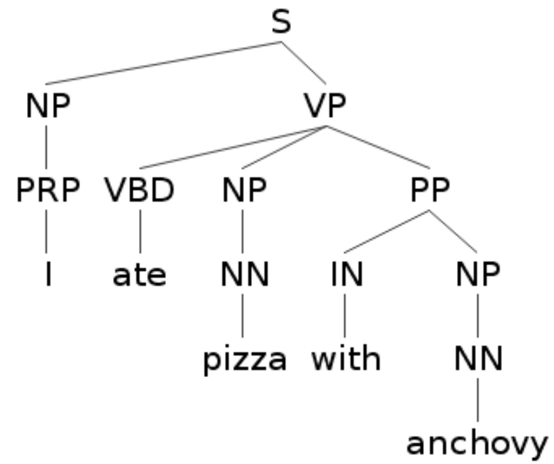


S → NP VP (1)

NP → PRP (1)

From Treebank to (P)CFG



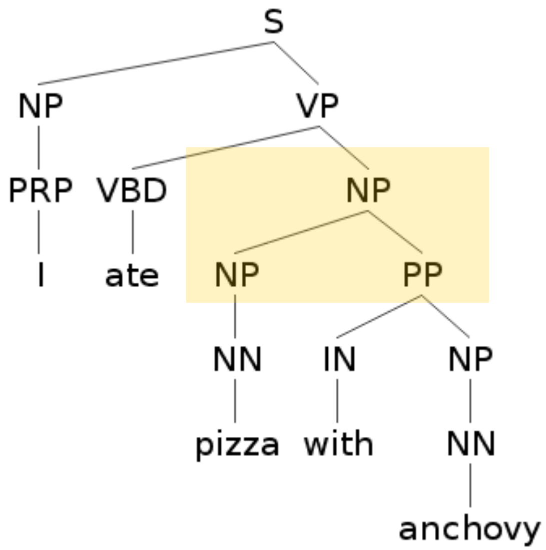


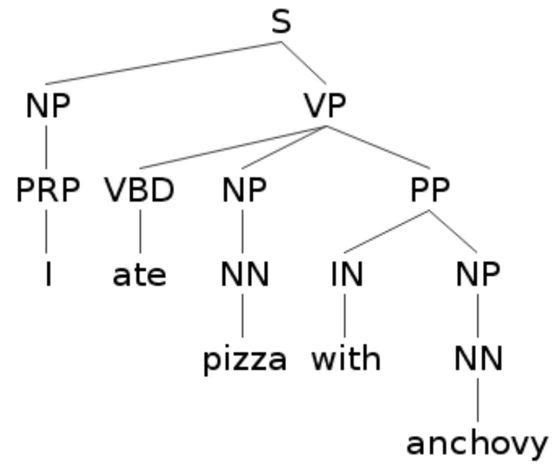
S → NP VP (1)

NP → PRP (1)

VP → VBD NP (1)

From Treebank to (P)CFG





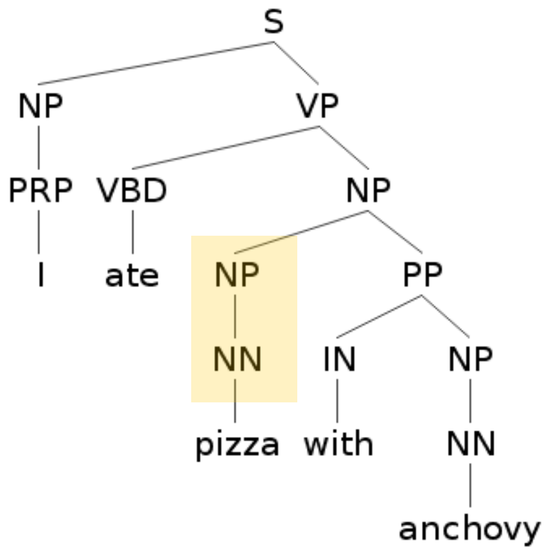
S → NP VP (1)

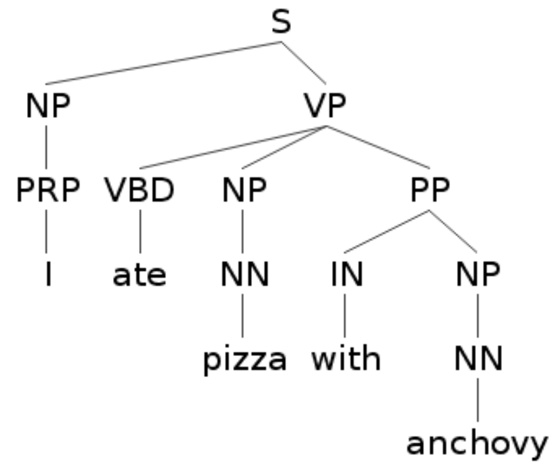
NP → PRP (1)

VP → VBD NP (1)

NP → NP PP (1)

From Treebank to (P)CFG





S → NP VP (1)

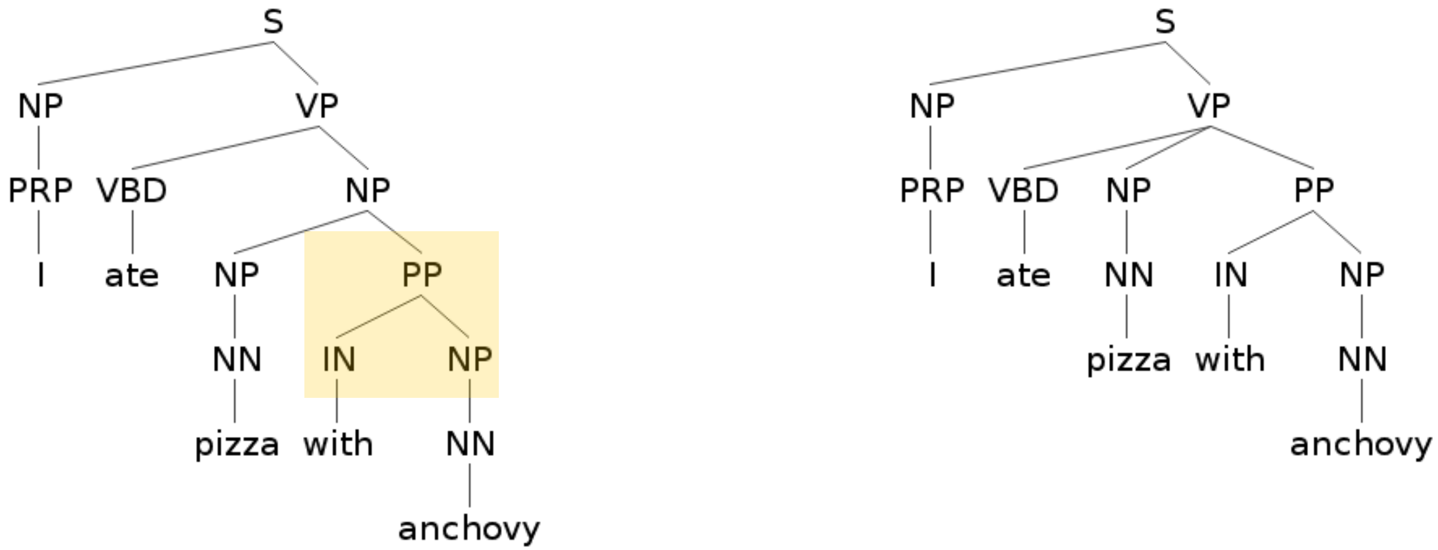
NP → PRP (1)

VP → VBD NP (1)

NP → NP PP (1)

NP → NN (1)

From Treebank to (P)CFG



S → NP VP (1)

NP → PRP (1)

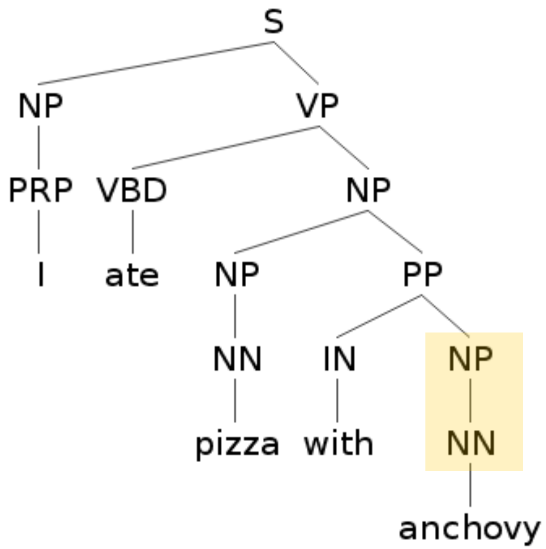
VP → VBD NP (1)

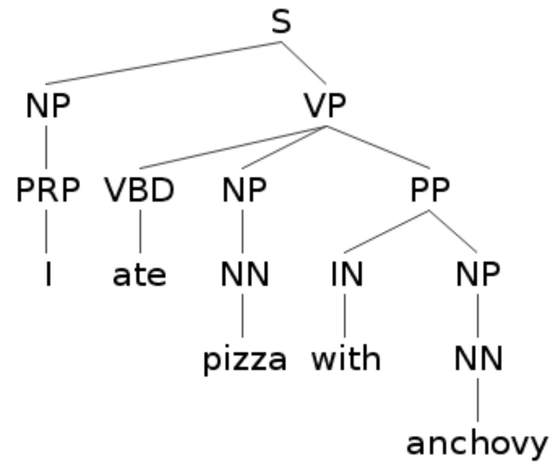
NP → NP PP (1)

NP → NN (1)

PP → IN NP (1)

From Treebank to (P)CFG





S → NP VP (1)

NP → PRP (1)

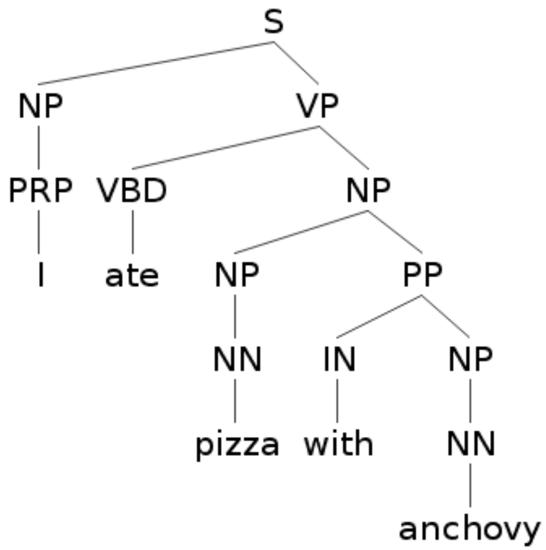
VP → VBD NP (1)

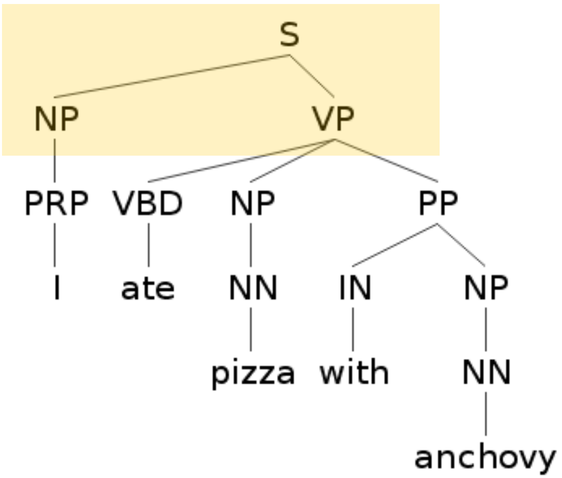
NP → NP PP (1)

NP → NN (2)

PP → IN NP (1)

From Treebank to (P)CFG





S → NP VP (2)

NP → PRP (1)

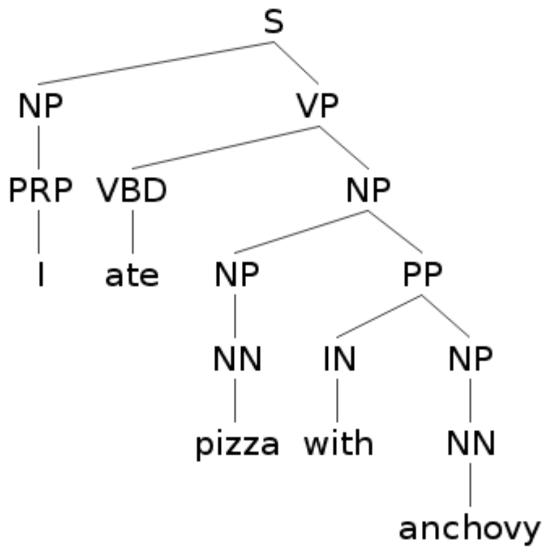
VP → VBD NP (1)

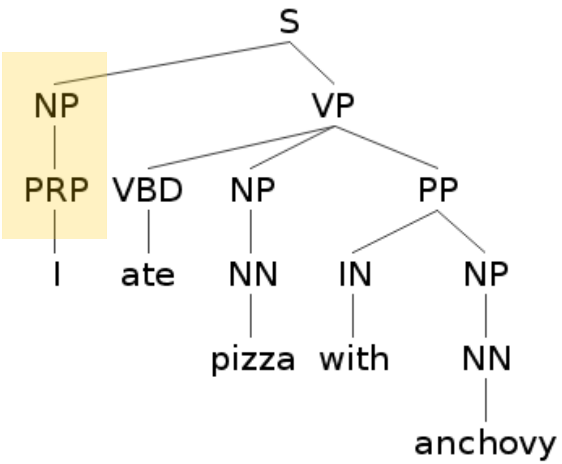
NP → NP PP (1)

NP → NN (2)

PP → IN NP (1)

From Treebank to (P)CFG





S → NP VP (2)

NP → PRP (2)

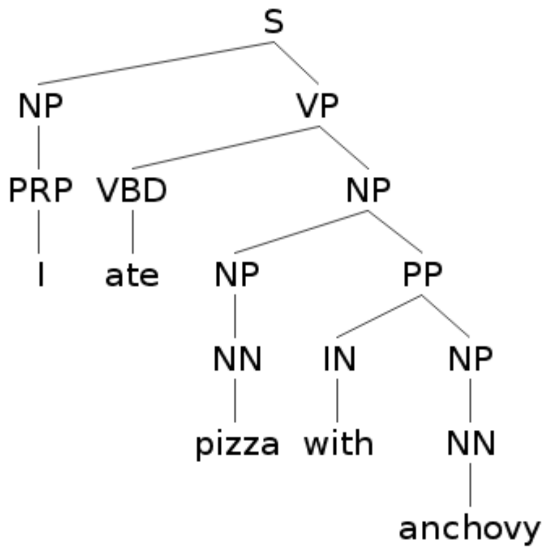
VP → VBD NP (1)

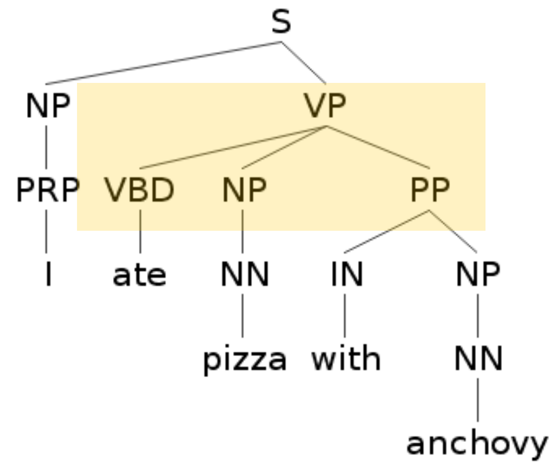
NP → NP PP (1)

NP → NN (2)

PP → IN NP (1)

From Treebank to (P)CFG





S → NP VP (2)

NP → PRP (2)

VP → VBD NP (1)

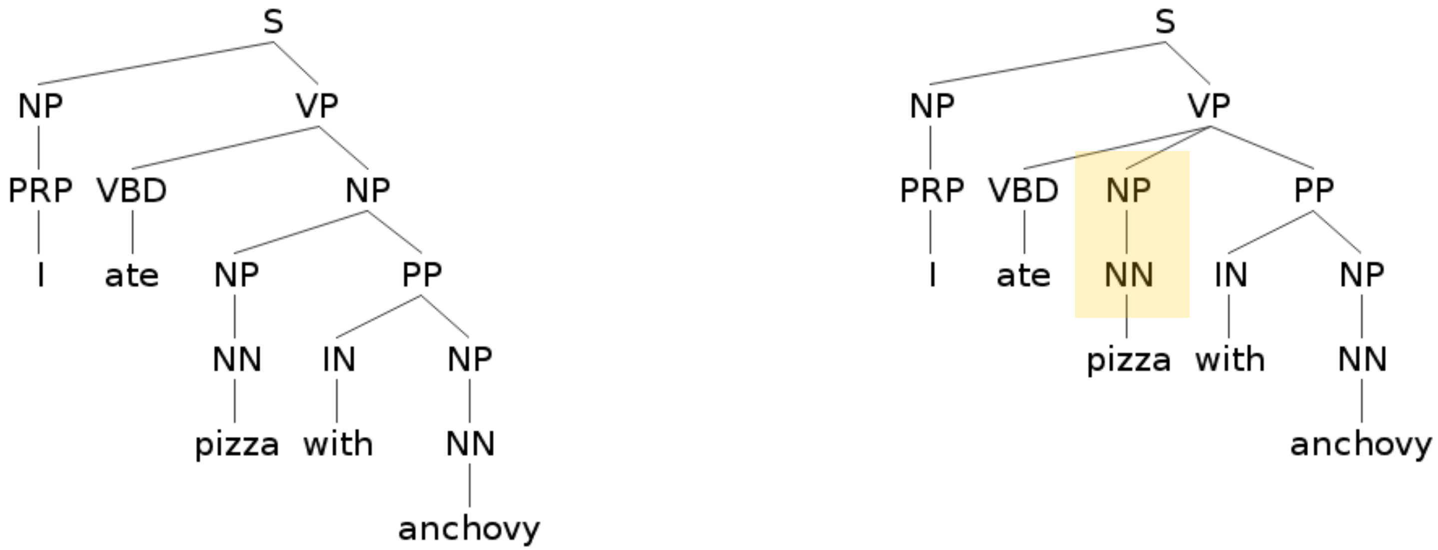
NP → NP PP (1)

NP → NN (2)

PP → IN NP (1)

VP → VBD NP PP (1)

From Treebank to (P)CFG



S → NP VP (2)

NP → PRP (2)

VP → VBD NP (1)

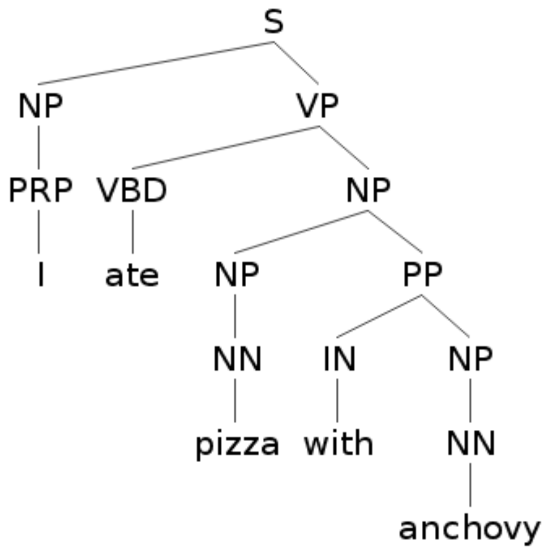
NP → NP PP (1)

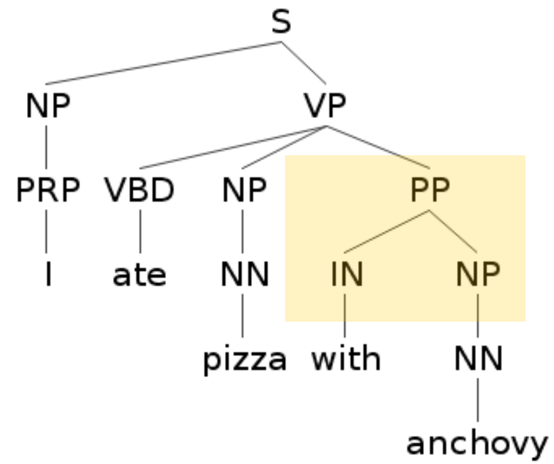
NP → NN (3)

PP → IN NP (1)

VP → VBD NP PP (1)

From Treebank to (P)CFG





S → NP VP (2)

NP → PRP (2)

VP → VBD NP (1)

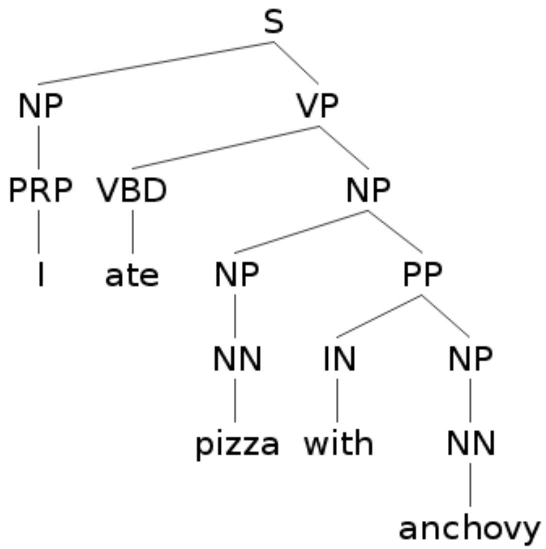
NP → NP PP (1)

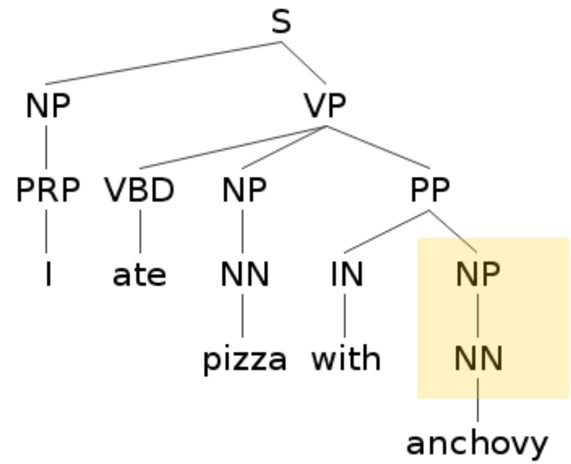
NP → NN (3)

PP → IN NP (2)

VP → VBD NP PP (1)

From Treebank to (P)CFG





S → NP VP (2)

NP → PRP (2)

VP → VBD NP (1)

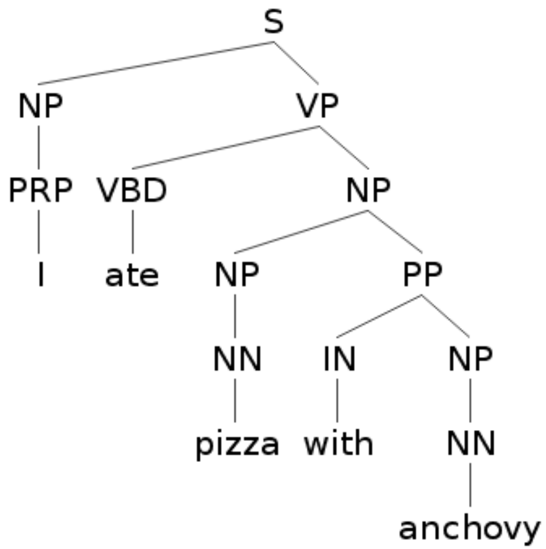
NP → NP PP (1)

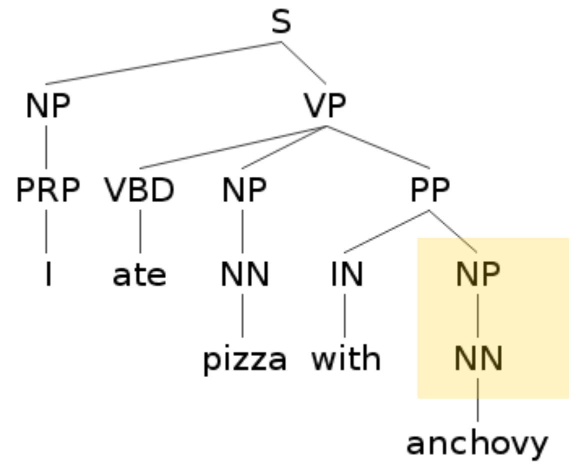
NP → NN (4)

PP → IN NP (2)

VP → VBD NP PP (1)

From Treebank to PCFG





S → NP VP (2/2)

NP → PRP (2/7)

VP → VBD NP (1/2)

NP → NP PP (1/7)

NP → NN (4/7)

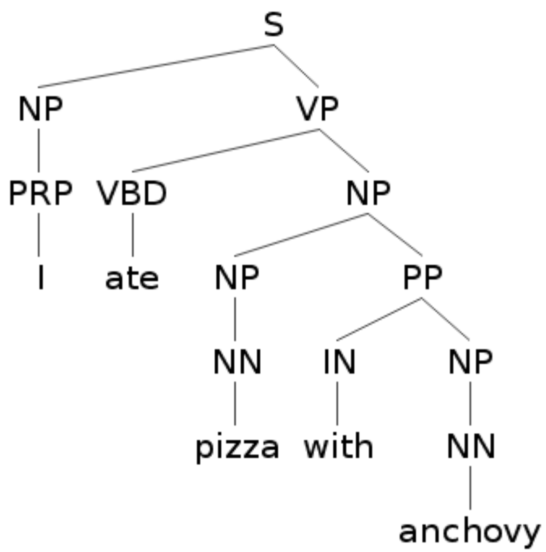
PP → IN NP (2/2)

VP → VBD NP PP (1/2)

Probability of a rule is its observed frequency divided by the total number of observed rules with the same non-terminal on the left-hand side

$$P(tree) = \prod P(rule_i)$$

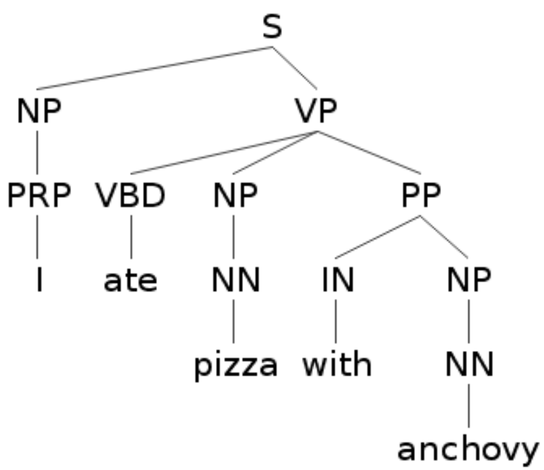
P(



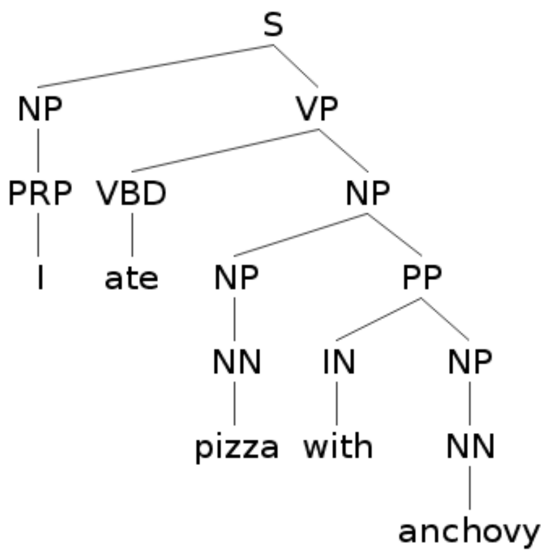
)

$$= 1*2/7*1/2*1/7*4/7*1*4/7$$
$$= 16/2401$$

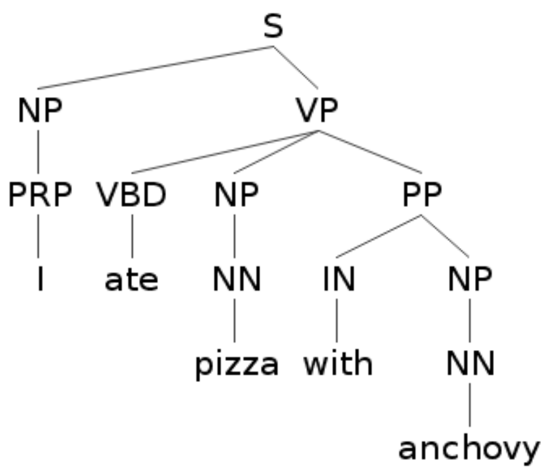
$P(\text{tree}) = \prod P(\text{rule}_i)$

P() $= 1*2/7*1/2*4/7*1*4/7$
$= 16/343$

$P(\text{tree}) = \prod P(\text{rule}_i)$



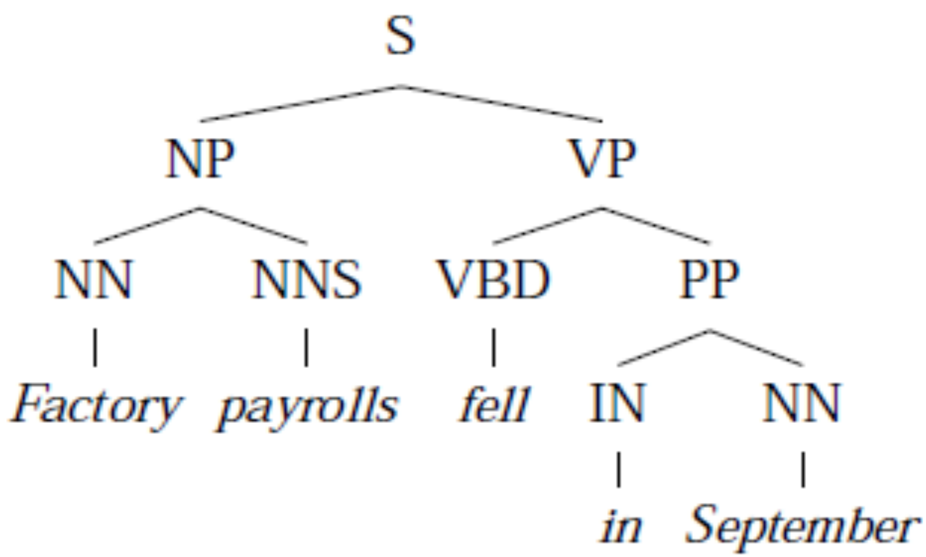
16/2401

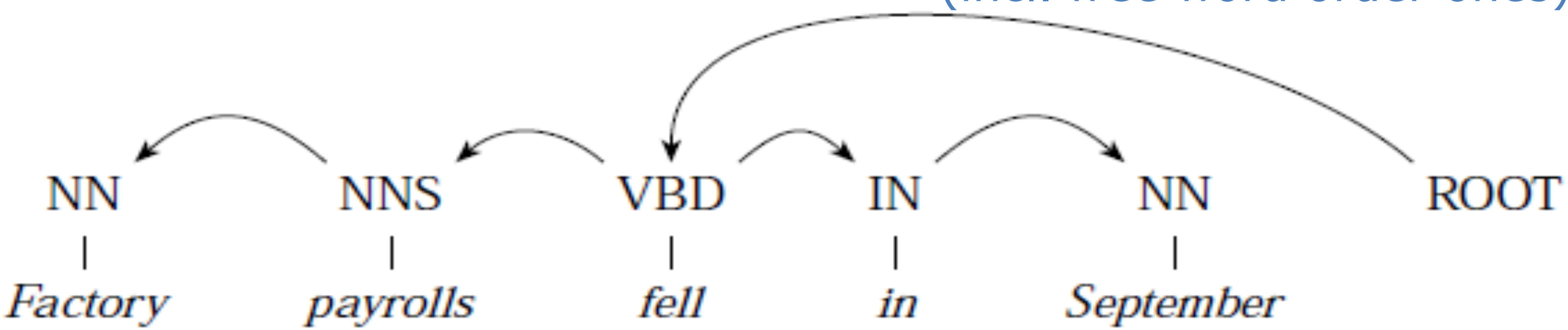


16/343

Dependency Grammars

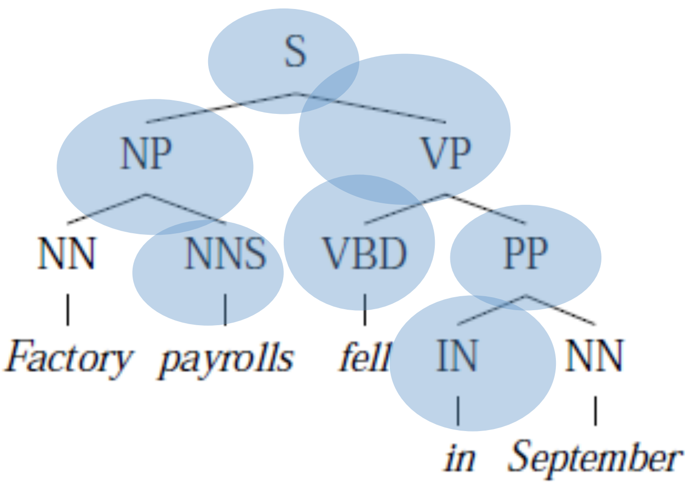
- Model of syntactic **links** between combinations of 2 words: 1 head, 1 dependent
- **Structural restrictions** on dependencies
- Typed dependencies attribute a **lexical-semantic value** to the link (modifier, subject, object, determiner, …)
- Well suited for all languages (incl. free word order ones)



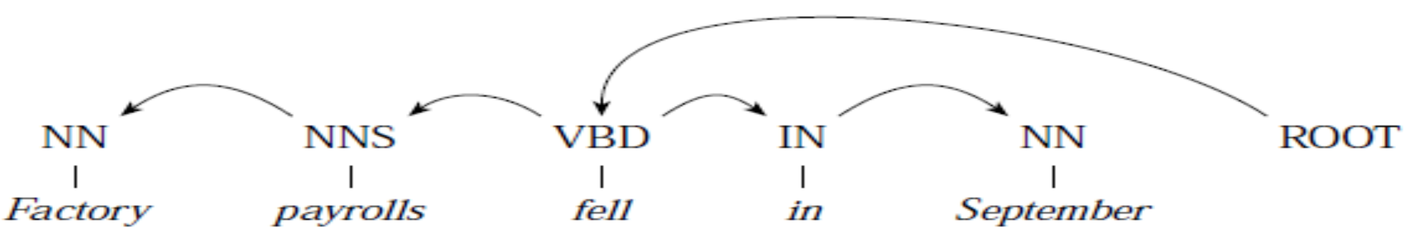


Dependency Grammars

- Using **head-dependency rules**, you can transform a typical tree structures into an (unlabeled) dependency graph automatically.

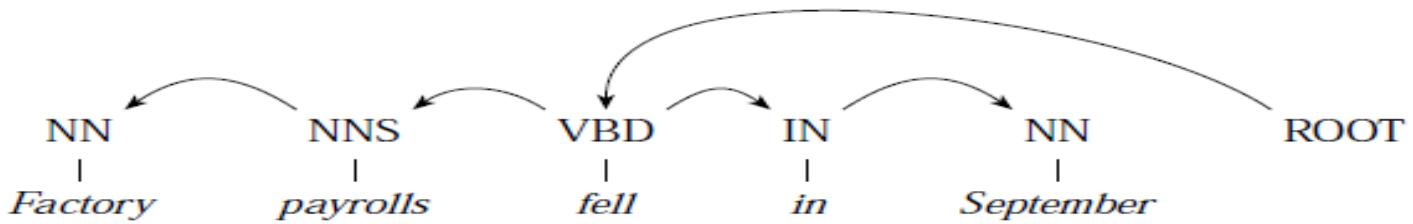


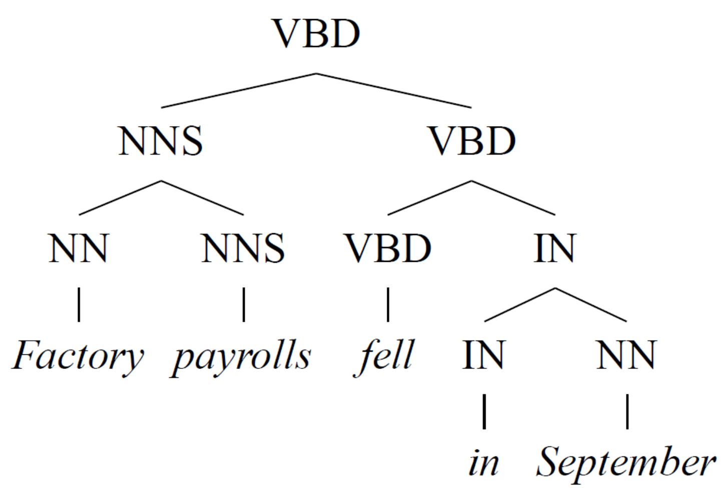
- Rules:
  1. Mark head-child of each node in tree structure (head percolation table)
  2. In dependency structure, make head of each non-child depend on the head of the head-child
- The Penn Treebank becomes a dependency graph bank



Dependency Grammars

- From dependency graph to phrase structure tree





Shallow Parsing

- For many NLP applications we don’t need a full parse tree (e.g. information extraction)
- **Shallow Parsing** identifies syntactic chunks (*X*P chunking) in a sentence and *shallow* flags such as SBJ and OBJ markers
- Also sometimes called Partial Parsing

[NP-SBJ The dog] [VP chases] [NP-OBJ the cat] [ADVP all day long] [ADVP now] though

Shallow Parsing

[NP-SBJ The dog] [VP chases] [NP-OBJ the cat] [ADVP all day long] [ADVP now] though

- This can be done **faster, more accurately** than full parsing
- This can be done using part-of-speech tagging technology by assigning **IOB tags** (inside/outside/begin) to tokens in a sentence

The/DT/NP-I
dog/NN/NP-I
chases/VBZ/VP-I
the/DT/NP-I
cat/NN/NP-I
all/DT/ADVP-I
day/NN/ADVP-I
long/RB/ADVP-I
now/RB/ADVP-B
though/RB/O

Shallow Parsing

[NP-SBJ The dog] [VP chases] [NP-OBJ the cat] [ADVP all day long] [ADVP now] though

- You can add "**flags**" to the annotation scheme that cover additional syntactic roles you want to recognize
- IOB tagging is also used in named entity recognition

The/DT/NP-SBJ-I
dog/NN/NP-SBJ-I
chases/VBZ/VP-I
the/DT/NP-OBJ-I
cat/NN/NP-OBJ-I
all/DT/ADVP-I
day/NN/ADVP-I
long/RB/ADVP-I
now/RB/ADVP-B
though/RB/O

(Full) Parsing

Parser vs Grammar

GRAMMAR

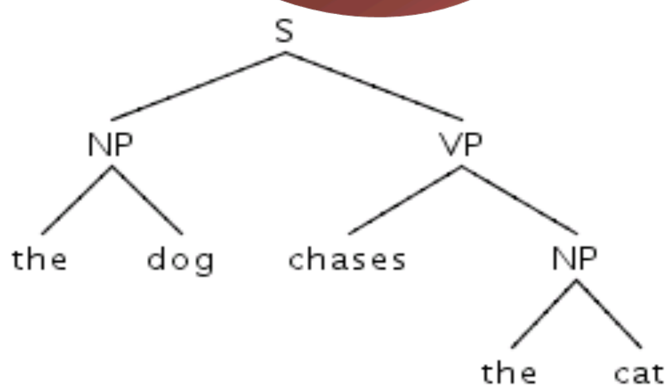
CFG,

dependency,

…

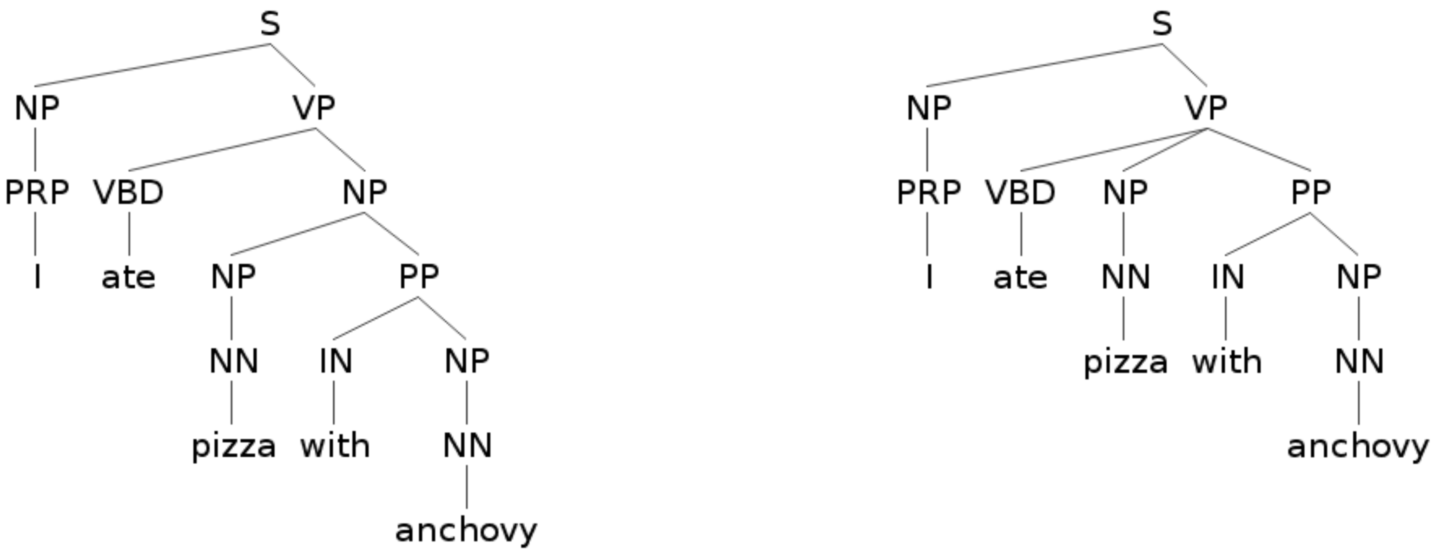
PARSER

top-down vs bottom-up, depth-first vs breadth-first, chart-parsing, Tomita



Generating a Parse Forest

- **Given** = **grammar**
- **Needed** = **parser**
- **Challenge** = **ambiguity**



Bottom-Up vs Top-Down

Bottom-up	Top-down
• Builds tree-structure from bottom of the structure to the top • Starts with the words • Grammar: – S ← NP VP – NP ← NNP – NP ← DT NN – VP ← V NP • Parse = successful ⇔ the parser can reach the top-level symbol at the root node of a structure that contains all the words of the sentence.	• Builds tree-structure from the top node down to the bottom • Starts with the top-node • Grammar: – S → NP VP – NP → NNP – NP → DT NN – VP → V NP • Parse = successful ⇔ the root-node heads a structure containing all the words of the sentence

Bottom-Up Parsing

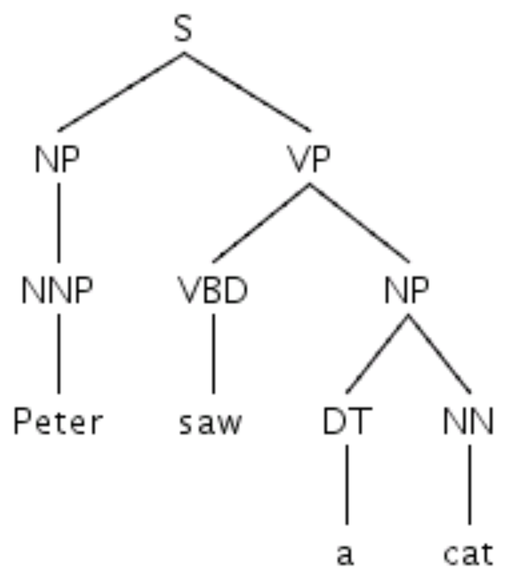
GRAMMAR
S → NP VP
NP → NNP
NP → DT NN
VP → VBD NP

LEXICON
NNP → Peter
VBD → saw
DT → a
NN → cat

but really

GRAMMAR
S ← NP VP
NP ← NNP
NP ← DT NN
VP ← VBD NP

LEXICON
NNP ← Peter
VBD ← saw
DT ← a
NN ← cat



- Parse for "*Peter saw a cat*", using this lexicon+grammar

Peter saw a cat	
NNP VBD DT NN	rewrite NP
NP VBD DT NN	rewrite NP
NP VBD NP	rewrite VP
NP VP	rewrite S
S	

- Parse = successful
  ⇔ the parser can reach the top-level symbol at the root node of a structure that contains all the words of the sentence.

Top-Down Parsing

GRAMMAR

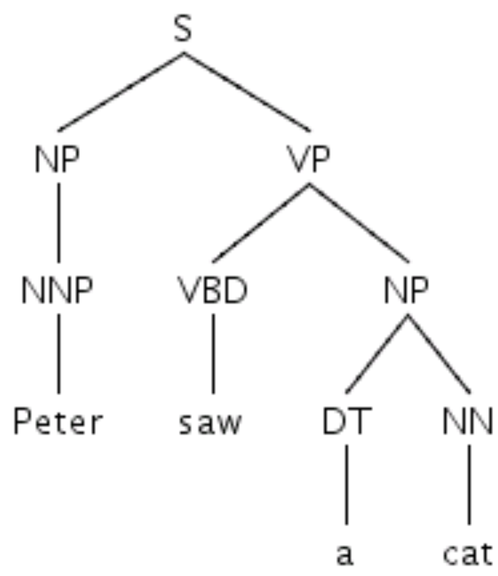
S → NP VP
NP → NNP
NP → DT NN
VP → VBD NP

LEXICON

NNP → Peter
VBD → saw
DT → a
NN → cat

Parse for "*Peter saw a cat*", using this lexicon+grammar

S	→ **NP VP**	rewrite NP
	→ **NNP VP**	rewrite NNP
	→ Peter **VP**	rewrite VP
	→ Peter **VBD NP**	rewrite VBD
	→ Peter saw **NP**	rewrite NP
	→ Peter saw **DT NN**	rewrite DT
	→ Peter saw a **NN**	rewrite NN
	→ Peter saw a cat	**SUCCESS**



- Parse = successful
  ⇔ the root-node heads a structure containing all the words of the sentence

Traversing the Search Space

- Parsing = traversing the search space of possible parses

- e.g. TOP-DOWN PARSING
  Parse for "*Peter saw a cat*", using this lexicon+grammar
  S → NP VP rewrite NP
  2 possibilities:
  → **NNP VP** rewrite NNP
  → DT NN VP rewrite DT

- Different search strategies (depth first/breadth first) to traverse the search space, but end-results stays the same
  = PARSE FOREST

Backtracking Top-Down Parsing Algorithm

- Initialize state-list (S 1) (*current_symbol position*)
  - Find rewrite rule in grammar with “S” as left-hand side symbol
  - No such rule: FAIL

- Take 1st state
  = empty
    - SUCCESS if position is equal to end of the sentence
    - otherwise FAIL

- Generate following states:
  - 1st symbol is lexical and next word belongs to this category
    - New state: shift to next symbol
    - Increment position
  - 1st symbol is non-lexical
    - Generate new state for each rule in the grammar with that symbol on the left-hand side
    - Add to state-list (depth1st vs breadth1st)

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5
S1

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5	
S1	
VP NP 1	NP VP 1

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5		
S1		
VP NP 1		NP VP 1
vbd NP1	vbd NP NP1	

Universiteit Antwerpen

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="3">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="3">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td>NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td></td></tr>
</table>

Depth-First

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5		
S1		
VP NP 1		NP VP 1
vbd NP1	vbd NP NP1	
FAIL		

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="3">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="3">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td>NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td rowspan="2"></td></tr>
<tr><td>FAIL</td><td></td></tr>
</table>

BACKTRACK

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="3">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="3">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td>NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td rowspan="2"></td></tr>
<tr><td>FAIL</td><td>FAIL</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="3">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="3">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td>NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td rowspan="2"></td></tr>
<tr><td>FAIL</td><td>FAIL</td></tr>
</table>

BACKTRACK

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5			
S1			
VP NP 1		NP VP 1	
vbd NP1	vbd NP NP1	dt nn VP 1	nnp VP 1
FAIL	**FAIL**		

Universiteit Antwerpen

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5			
S1			
VP NP 1		NP VP 1	
vbd NP1	vbd NP NP1	dt nn VP 1	nnp VP 1
FAIL	**FAIL**	**FAIL**	

.
.
.
.
.

Top-Down Depth-First (Backtracking)

Grammar
- S → NP VP
- S → VP NP
- NP → NNP
- NP → DT NN
- VP → VBD
- VP → VBD NP

Lexicon
- NNP → Peter
- VBD → saw|cried|is
- DT → a
- NN → cat

1 Peter 2 saw 3 a 4 cat 5			
S1			
VP NP 1		NP VP 1	
vbd NP1	vbd NP NP1	dt nn VP 1	nnp VP 1
FAIL	FAIL	FAIL	

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

1 Peter 2 saw 3 a 4 cat 5			
S1			
VP NP 1		NP VP 1	
vbd NP1	vbd NP NP1	dt nn VP 1	nnp VP 1
FAIL	**FAIL**	**FAIL**	

.
.
.
.
.

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="4">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="4">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="2">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td>nnp VP 1</td></tr>
<tr><td>FAIL</td><td>FAIL</td><td>FAIL</td><td>VP 2</td></tr>
</table>

Top-Down Depth-First (Backtracking)

<u>Grammar</u>
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
<u>Lexicon</u>
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="5">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="5">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="3">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="2">nnp VP 1</td></tr>
<tr><td rowspan="2">FAIL</td><td rowspan="2">FAIL</td><td rowspan="2">FAIL</td><td colspan="2">VP 2</td></tr>
<tr><td>vbd 2</td><td>vbd NP 2</td></tr>
</table>

.
.
.

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="5">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="5">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="3">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="2">nnp VP 1</td></tr>
<tr><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td colspan="2">VP 2</td></tr>
<tr><td>vbd 2</td><td>vbd NP 2</td></tr>
<tr><td>() 3</td><td rowspan="2"></td></tr>
<tr><td></td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="5">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="5">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="3">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="2">nnp VP 1</td></tr>
<tr><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td colspan="2">VP 2</td></tr>
<tr><td>vbd 2</td><td>vbd NP 2</td></tr>
<tr><td>() 3</td><td rowspan="2"></td></tr>
<tr><td>FAIL</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="5">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="5">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="3">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="2">nnp VP 1</td></tr>
<tr><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td colspan="2">VP 2</td></tr>
<tr><td>vbd 2</td><td>vbd NP 2</td></tr>
<tr><td>() 3</td><td rowspan="2"></td></tr>
<tr><td>FAIL</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="5">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="5">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="3">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="2">nnp VP 1</td></tr>
<tr><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td colspan="2">VP 2</td></tr>
<tr><td>vbd 2</td><td>vbd NP 2</td></tr>
<tr><td>() 3</td><td>NP 3</td></tr>
<tr><td>FAIL</td><td></td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
- S → NP VP
- S → VP NP
- NP → NNP
- NP → DT NN
- VP → VBD
- VP → VBD NP

Lexicon
- NNP → Peter
- VBD → saw|cried|is
- DT → a
- NN → cat

<table>
<tr><td colspan="6">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="6">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="4">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="3">nnp VP 1</td></tr>
<tr><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td rowspan="4">FAIL</td><td colspan="3">VP 2</td></tr>
<tr><td>vbd 2</td><td colspan="2">vbd NP 2</td></tr>
<tr><td>() 3</td><td colspan="2">NP 3</td></tr>
<tr><td>FAIL</td><td>dt nn 3</td><td>nnp 3</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="6">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="6">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="4">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="3">nnp VP 1</td></tr>
<tr><td rowspan="5">FAIL</td><td rowspan="5">FAIL</td><td rowspan="5">FAIL</td><td colspan="3">VP 2</td></tr>
<tr><td>vbd 2</td><td colspan="2">vbd NP 2</td></tr>
<tr><td>() 3</td><td colspan="2">NP 3</td></tr>
<tr><td rowspan="2">FAIL</td><td>dt nn 3</td><td>nnp 3</td></tr>
<tr><td>nn 4</td><td></td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="6">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="6">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="4">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="3">nnp VP 1</td></tr>
<tr><td rowspan="6">FAIL</td><td rowspan="6">FAIL</td><td rowspan="6">FAIL</td><td colspan="3">VP 2</td></tr>
<tr><td>vbd 2</td><td colspan="2">vbd NP 2</td></tr>
<tr><td>() 3</td><td colspan="2">NP 3</td></tr>
<tr><td rowspan="3">FAIL</td><td>dt nn 3</td><td>nnp 3</td></tr>
<tr><td>nn 4</td><td rowspan="2"></td></tr>
<tr><td>() 5</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="6">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="6">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="4">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="3">nnp VP 1</td></tr>
<tr><td rowspan="7">FAIL</td><td rowspan="7">FAIL</td><td rowspan="7">FAIL</td><td colspan="3">VP 2</td></tr>
<tr><td>vbd 2</td><td colspan="2">vbd NP 2</td></tr>
<tr><td>() 3</td><td colspan="2">NP 3</td></tr>
<tr><td rowspan="4">FAIL</td><td>dt nn 3</td><td>nnp 3</td></tr>
<tr><td>nn 4</td><td rowspan="3"></td></tr>
<tr><td>() 5</td></tr>
<tr><td>SUCCESS</td></tr>
</table>

Top-Down Depth-First (Backtracking)

Grammar
S → NP VP
S → VP NP
NP → NNP
NP → DT NN
VP → VBD
VP → VBD NP
Lexicon
NNP → Peter
VBD → saw|cried|is
DT → a
NN → cat

<table>
<tr><td colspan="6">1 Peter 2 saw 3 a 4 cat 5</td></tr>
<tr><td colspan="6">S1</td></tr>
<tr><td colspan="2">VP NP 1</td><td colspan="4">NP VP 1</td></tr>
<tr><td>vbd NP1</td><td>vbd NP NP1</td><td>dt nn VP 1</td><td colspan="3">nnp VP 1</td></tr>
<tr><td rowspan="7">FAIL</td><td rowspan="7">FAIL</td><td rowspan="7">FAIL</td><td colspan="3">VP 2</td></tr>
<tr><td>vbd 2</td><td colspan="2">vbd NP 2</td></tr>
<tr><td>() 3</td><td colspan="2">NP 3</td></tr>
<tr><td rowspan="4">FAIL</td><td>dt nn 3</td><td>nnp 3</td></tr>
<tr><td>nn 4</td><td rowspan="3">FAIL</td></tr>
<tr><td>() 5</td></tr>
<tr><td>SUCCESS</td></tr>
</table>

Problem of left-recursion

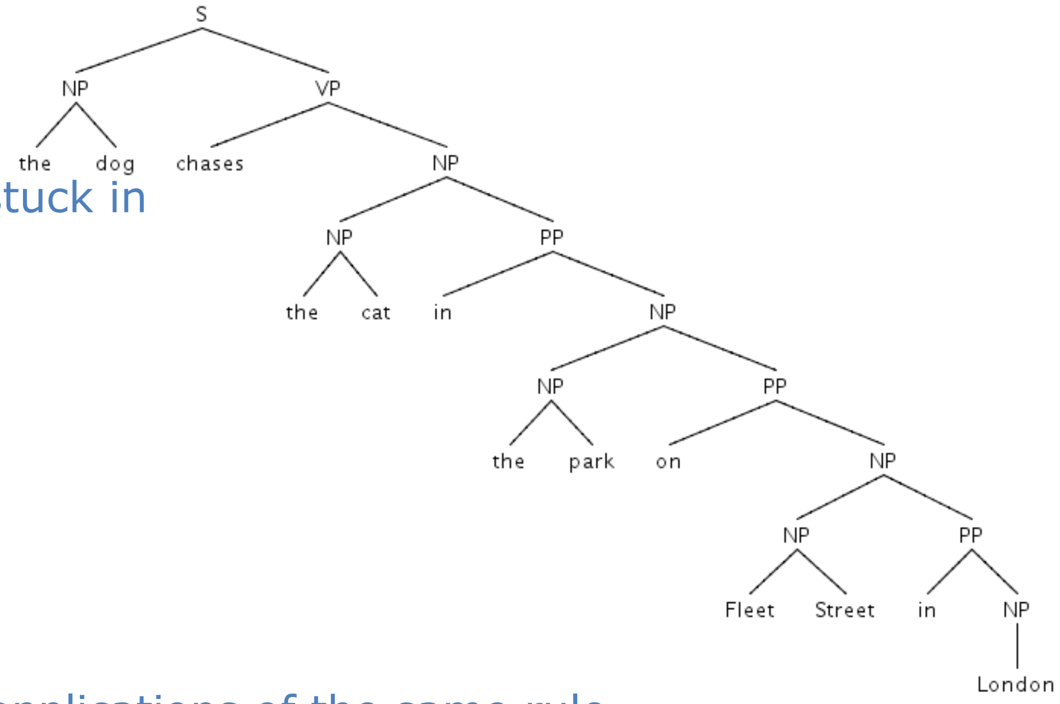
- **Recursion**: inherent to language

vb *the car in the back of the house in the street behind…*
or…

- Rules like

NP → NP PP

can cause a parser to get stuck in an infinite loop



- **Solutions**:
  - Adjust grammar

    NP → NP' PP
  - Adjust parser

    limit consecutive applications of the same rule

Problem of simple top-down/ bottom-up parsing

- Lot of redundant work
- Does not keep track of search operations that were already performed
- Solution: chart parsing

- **Chart-Parsing**: maintain matches that were found in a *chart.* This chart contains:
  - The words and their respective positions

    = initial **key list**
  - Rules of which 1 or more parts have already been matched on the input, but that are still not completely matched

    = *active arcs* (**agenda**)
  - Rules in which all elements have been matched (= constituents found)

    = *inactive arcs* (are added to **key list**)

Chart Parsing Algorithm

- Take the next element **C** (from position **p1** to **p2**) from the **key list**
- There exists a rule **r** in the grammar that starts with element **C**
  - ⇒ make an **active arc** for rule **r** from position **p1** to **p2** and put it in the agenda
- There exists an active arc in the agenda from position **p0** to **p1**, in which the next element to be matched is element **C**
  - ⇒ make a new active arc from position **p0** to **p2**, with the updated rule and put it in the agenda
- If one of the newly made arcs contains a completely matched rule, add the constituent category and its respective positions to the **key list**

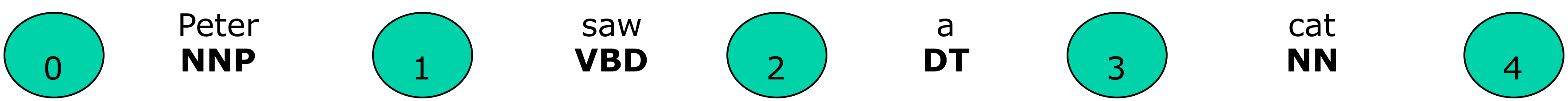
Chart Parsing

Keylist 0 Peter 1 saw 2 a 3 cat 4

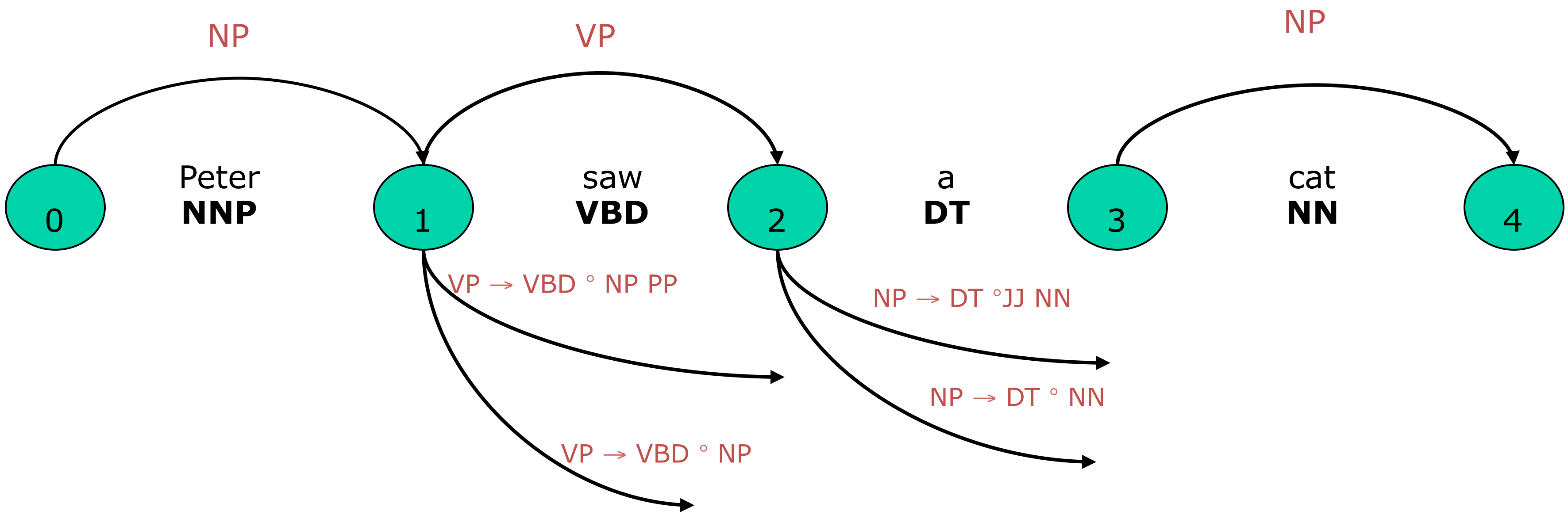
Grammar

S → NP VP
S → NP
NP → NNP
NP → NN
NP → DT JJ NN
NP → DT NN
NP → NP PP
VP → VBD NP
VP → VBD NP PP
VP → VBD
PP -> IN NP

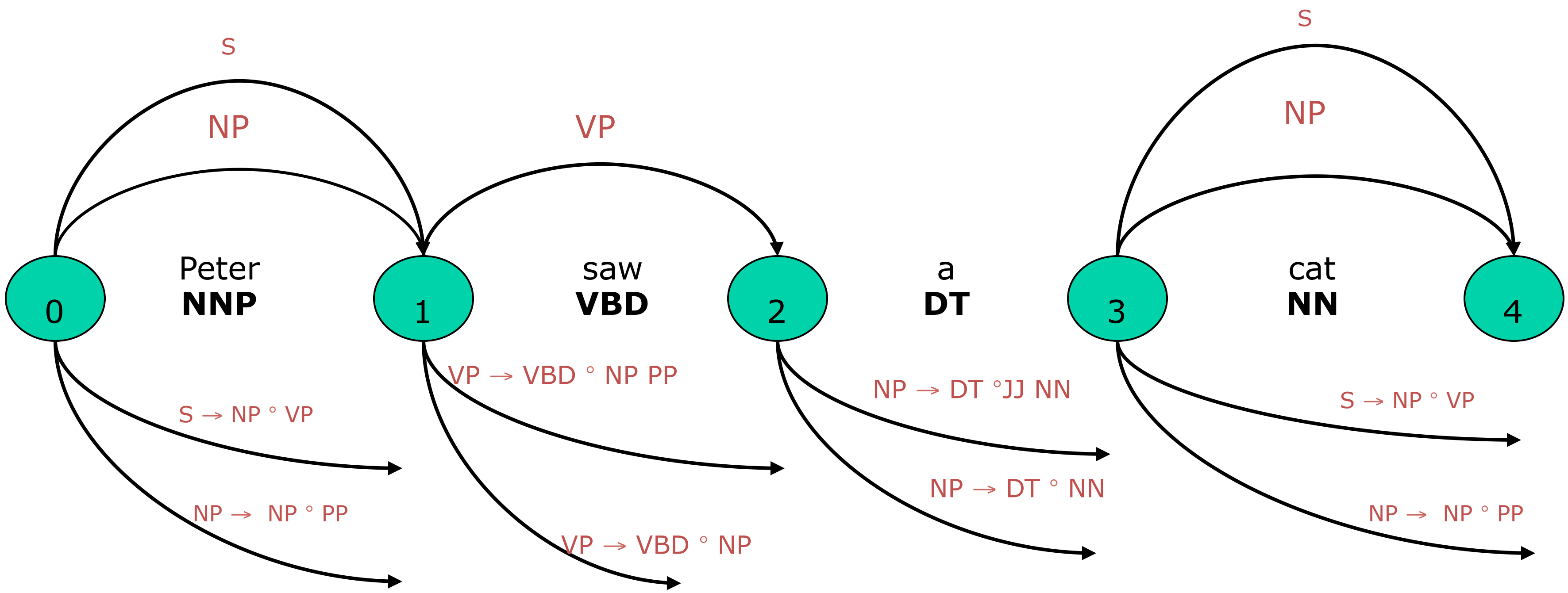
- Step 0: pos tag the keylist
  0 NNP 1 VBD 2 DT 3 NN 4
- Step 1: initialize agendas from key-list
- Step 2: repeat CP algorithm until no new arcs are made


0
Peter
NNP
1
saw
VBD
2
a
DT
3
cat
NN
4

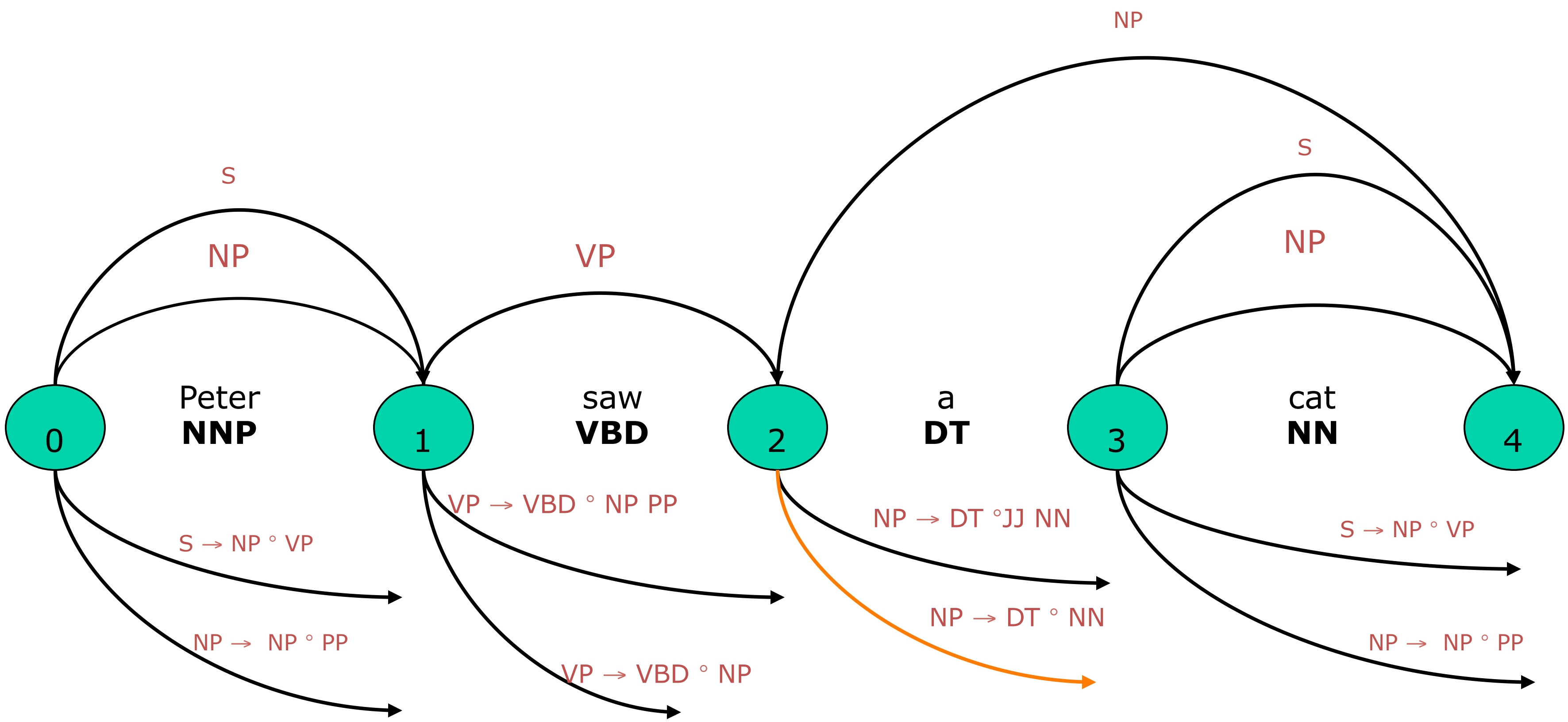
→ 0	1	NP → NNP °
→ 1	2	VP → VBD °
→ 1	2	VP → VBD ° NP PP
→ 1	2	VP → VBD ° NP
→ 2	3	NP → DT ° JJ NN
→ 2	3	NP → DT ° NN
→ 3	4	NP → NN °



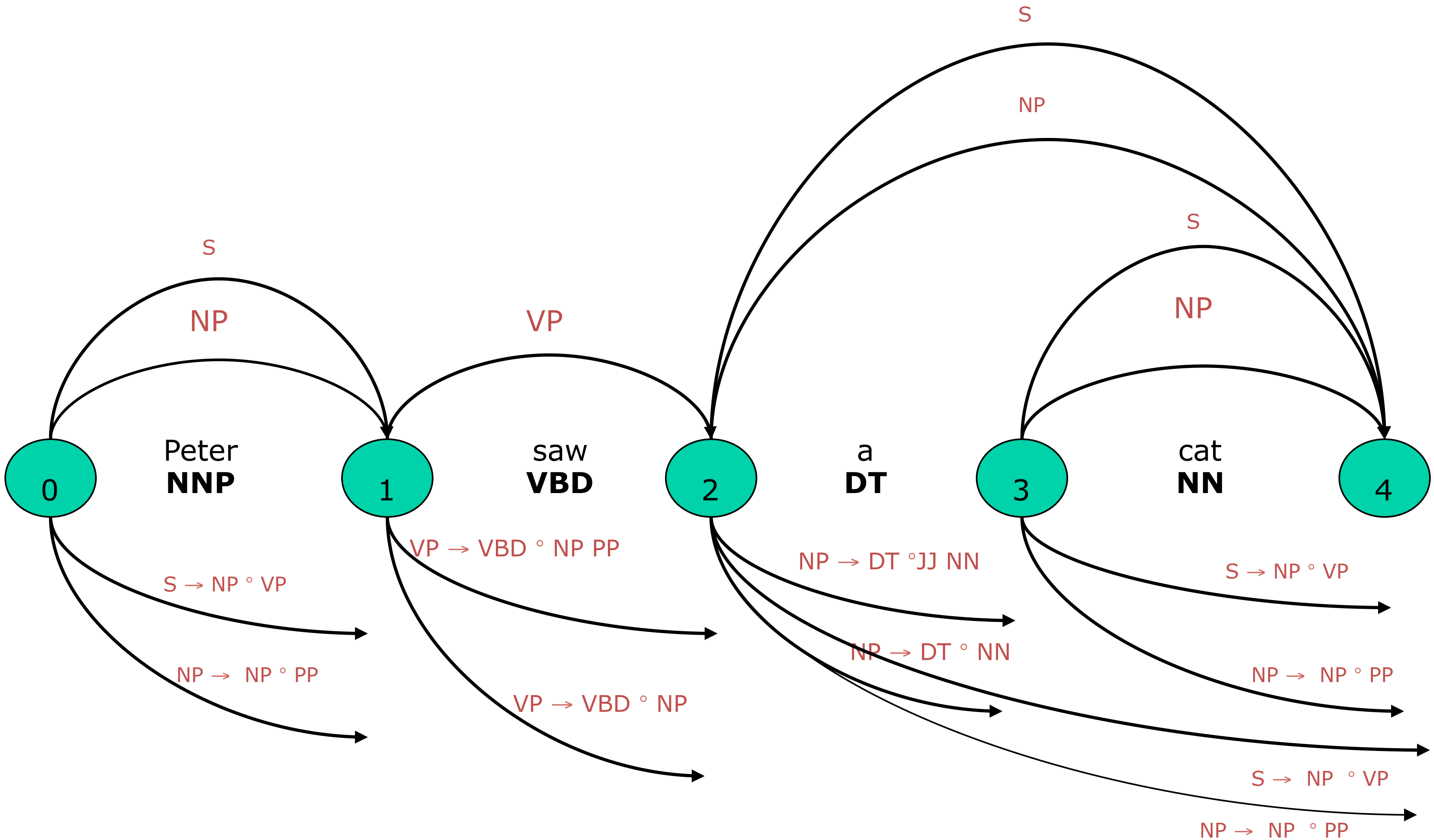
0	1	S → NP °
0	1	S → NP ° VP
0	1	NP → NP ° PP
3	4	S → NP °
3	4	S → NP ° VP
3	4	NP → NP ° PP
2	4	NP → DT NN °



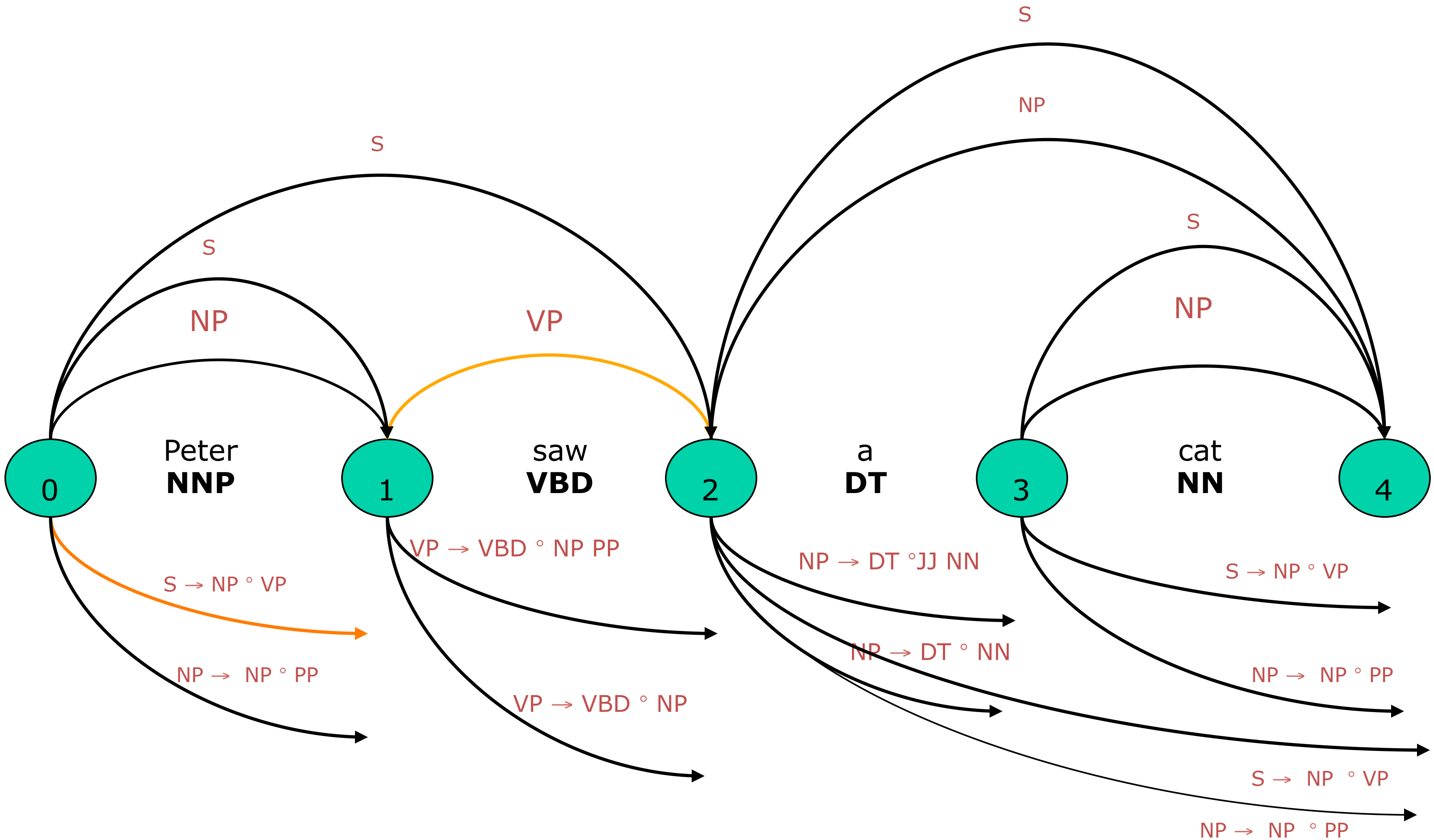
→ 0	1	S → NP °
→ 0	1	S → NP ° VP
→ 0	1	NP → NP ° PP
→ 3	4	S → NP °
→ 3	4	S → NP ° VP
→ 3	4	NP → NP ° PP
→ 2	4	NP → DT NN °



→	2	4	S → NP °
→	2	4	S → NP ° VP
→	2	4	NP → NP ° PP
	0	2	S → NP VP °
	1	4	VP → VBD NP ° PP
	1	4	VP → VBD NP °

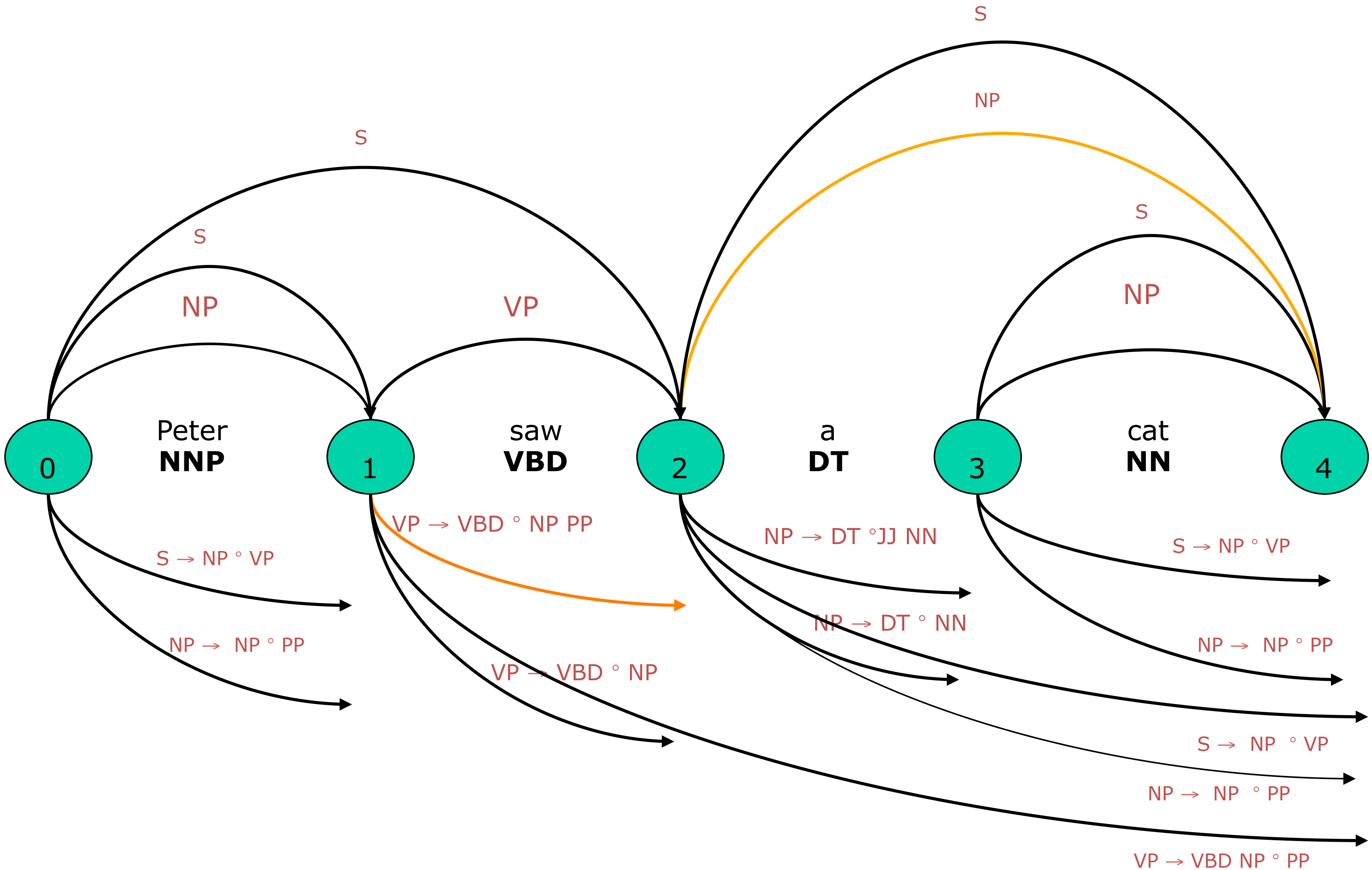


2	4	S → NP °
2	4	S → NP ° VP
2	4	NP → NP ° PP
→ 0	2	S → NP VP °
1	4	VP → VBD NP ° PP
1	4	VP → VBD NP °

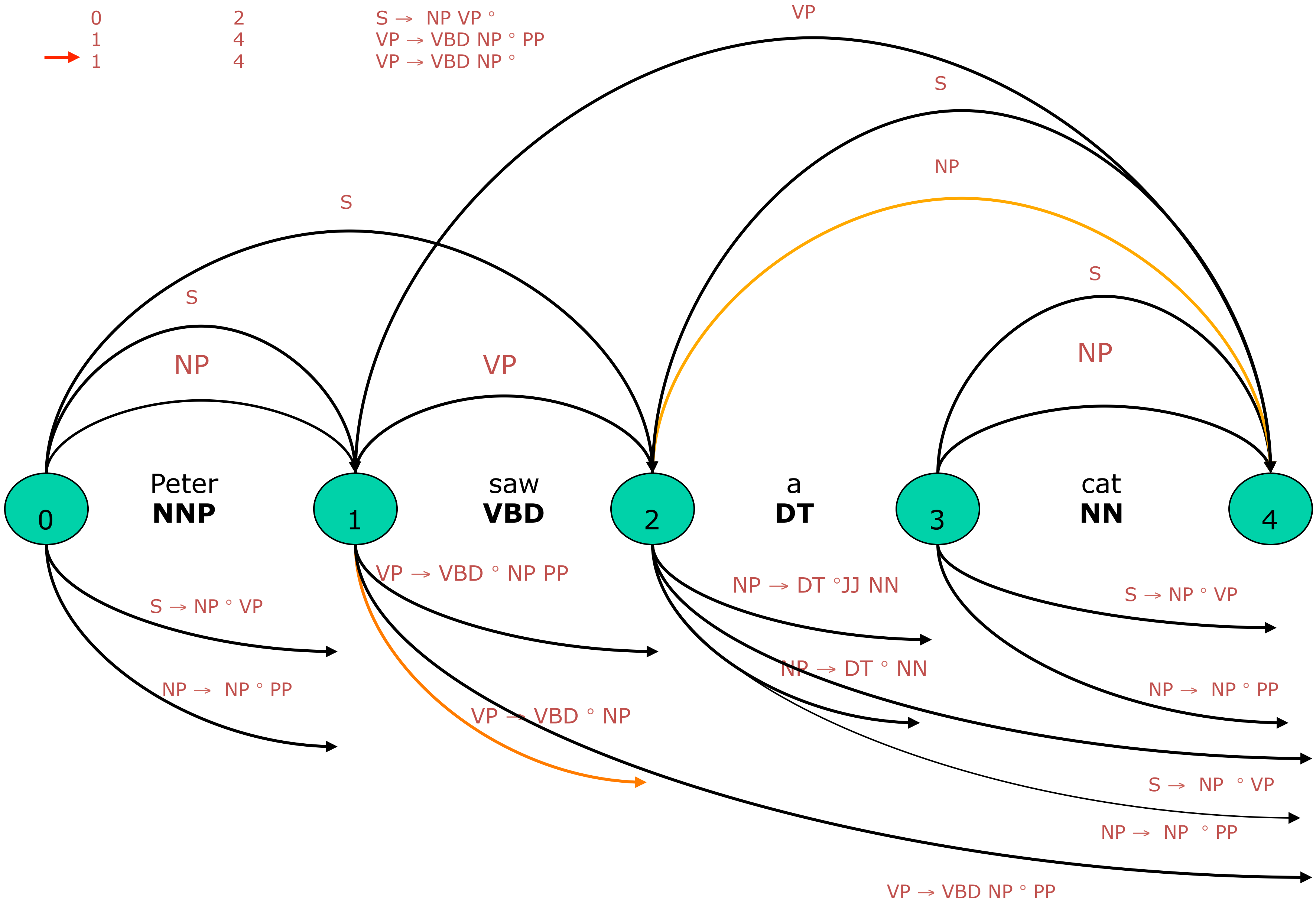


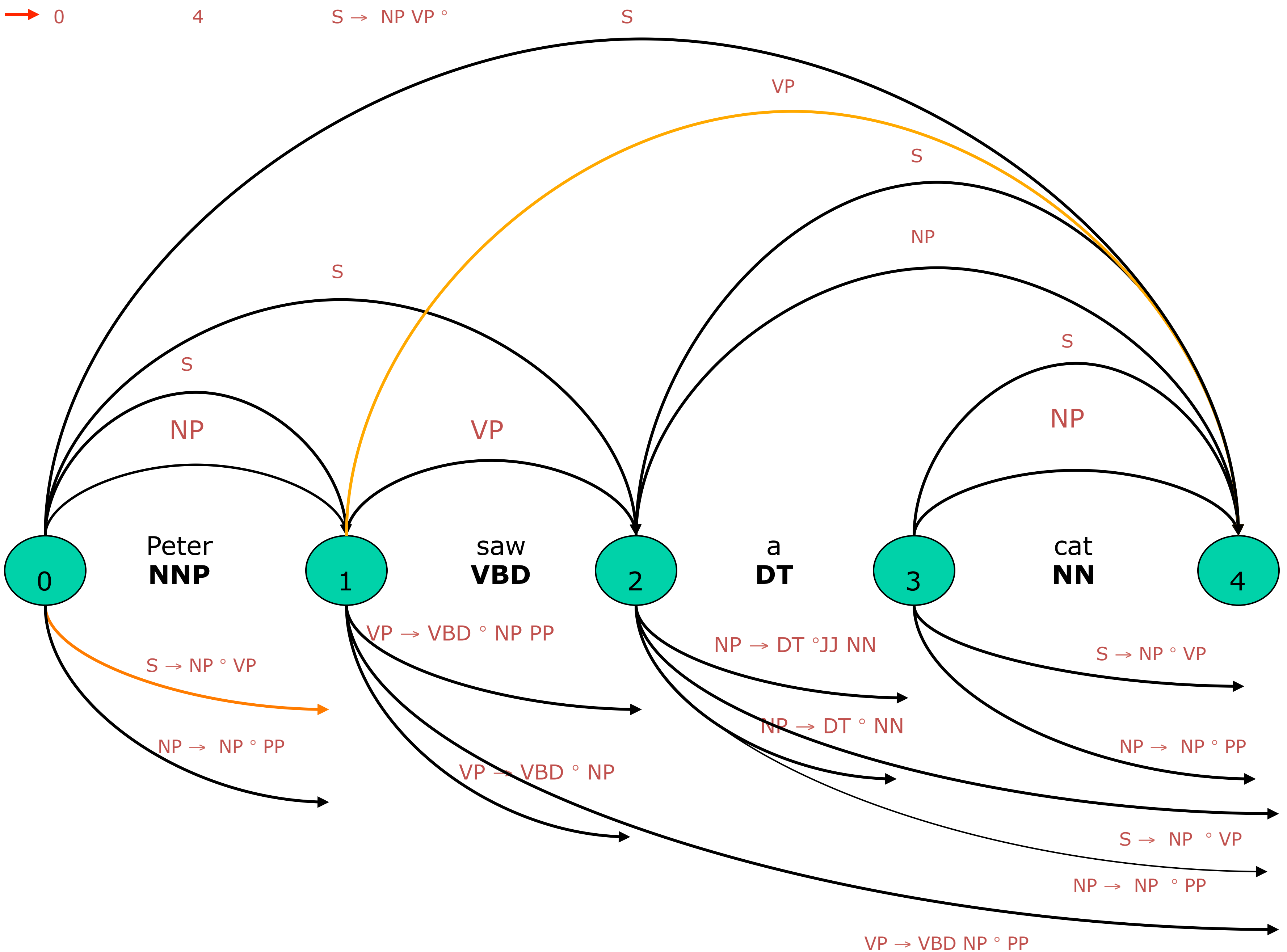
Universiteit Antwerpen

	2	4	S → NP °
	2	4	S → NP ° VP
	2	4	NP → NP ° PP
	0	2	S → NP VP °
→	1	4	VP → VBD NP ° PP
	1	4	VP → VBD NP °

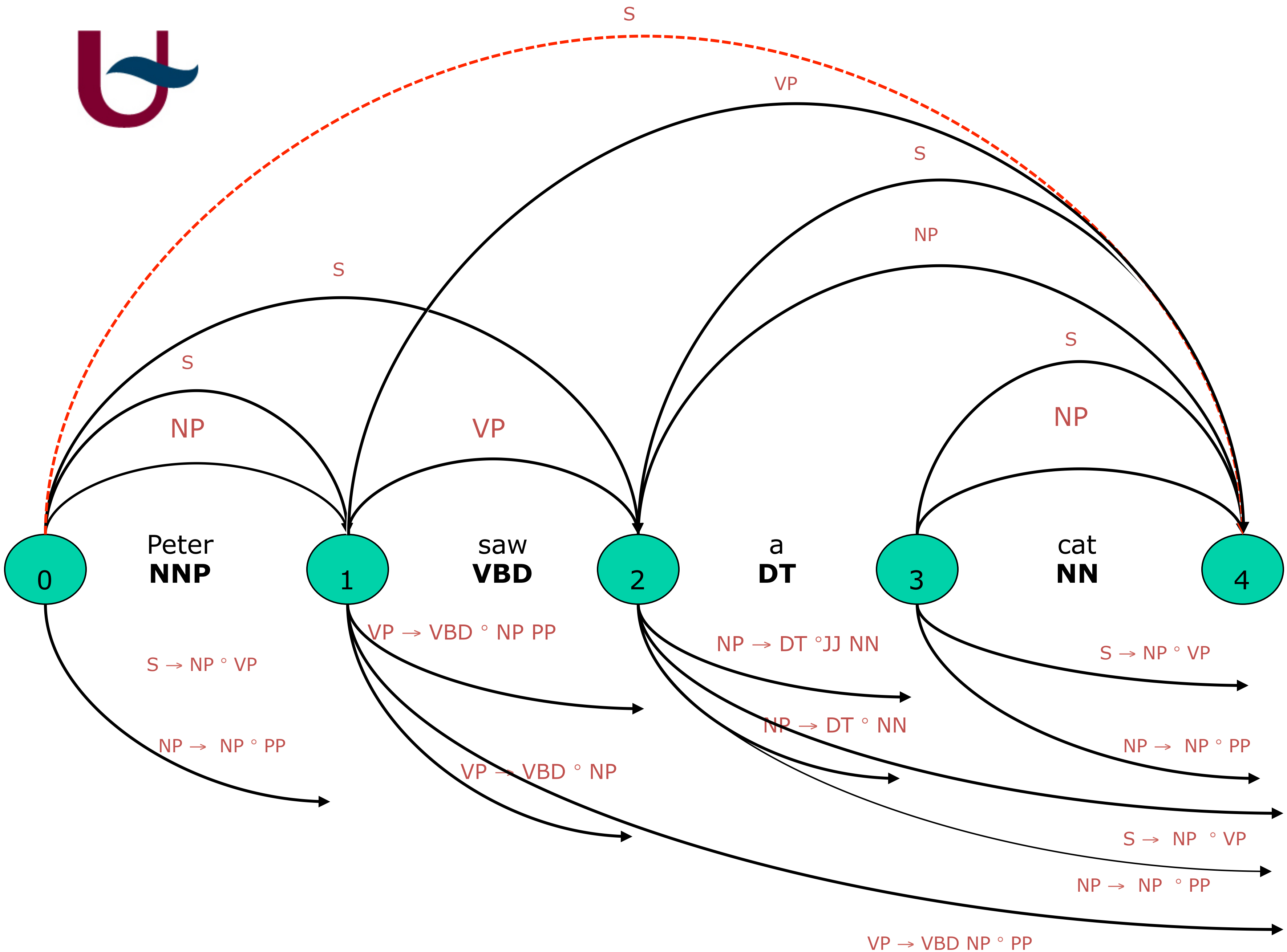


	2	4	S → NP °
	2	4	S → NP ° VP
	2	4	NP → NP ° PP
	0	2	S → NP VP °
	1	4	VP → VBD NP ° PP
→	1	4	VP → VBD NP °



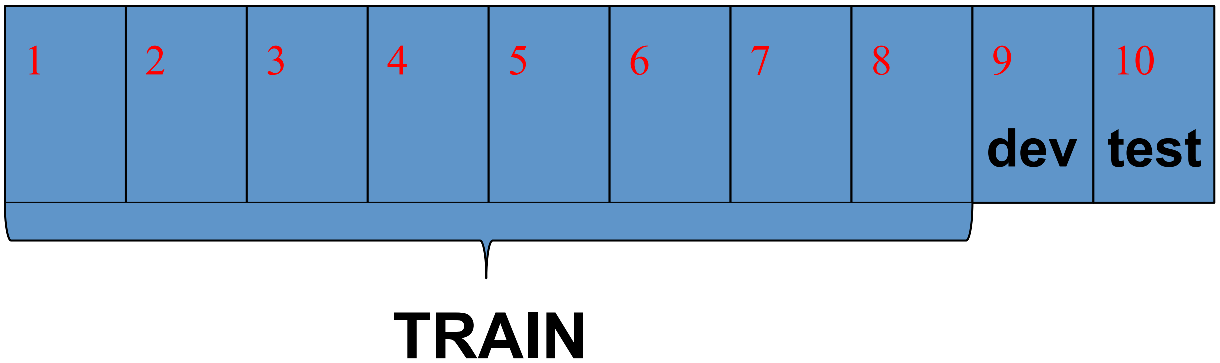


0
4
S → NP VP °
S
VP
S
NP
S
S
S
NP
VP
NP
0
Peter
NNP
1
saw
VBD
2
a
DT
3
cat
NN
4
VP → VBD ° NP PP
NP → DT °JJ NN
S → NP ° VP
S → NP ° VP
NP → DT ° NN
NP → NP ° PP
NP → NP ° PP
VP → VBD ° NP
S → NP ° VP
NP → NP ° PP
VP → VBD NP ° PP


S
VP
S
NP
S
S
S
NP
VP
NP
Peter
NNP
saw
VBD
a
DT
cat
NN
0
1
2
3
4
VP → VBD ° NP PP
S → NP ° VP
NP → DT °JJ NN
S → NP ° VP
NP → DT ° NN
NP → NP ° PP
VP → VBD ° NP
NP → NP ° PP
S → NP ° VP
NP → NP ° PP
VP → VBD NP ° PP

Evaluation

- Compare output of parser with *gold-standard* in treebank



- Score
  - Complete match of tree
  - Precision/Recall/F-Score on constituent level

Precision/Recall

- Cf. Information retrieval

Case study

100 documents all containing the word *queen*

50 (rock band) 40 (Elisabeth) 10(Beatrix)

- I want to know more about the rock band Queen

  Search Term: *Queen*

  100 documents returned *(i.e. all docs containing the word queen)*

Precision = correct / total returned = 50/100 = 0.5
Recall = correct / total in gold-standard = 50/50 = 1

Precision/Recall

- Cf. Information retrieval

Case study

Search Terms: *Queen Freddie*

60 documents returned (i.e. containing the words *Queen* AND *Freddie*)

- 45 on rock band Queen (5 rock docs didn't mention *Freddie*)
- 15 on Queen Elisabeth (that happen to contain the word *Freddie*)
- 2 on Queen Beatrix (that happen to contain the word *Freddie*)

Precision	= correct / total returned	= 45/60 = 0.75
Recall	= correct / total in gold-standard	= 45/50 = 0.9

F-Score

- Harmonic mean of precision & recall

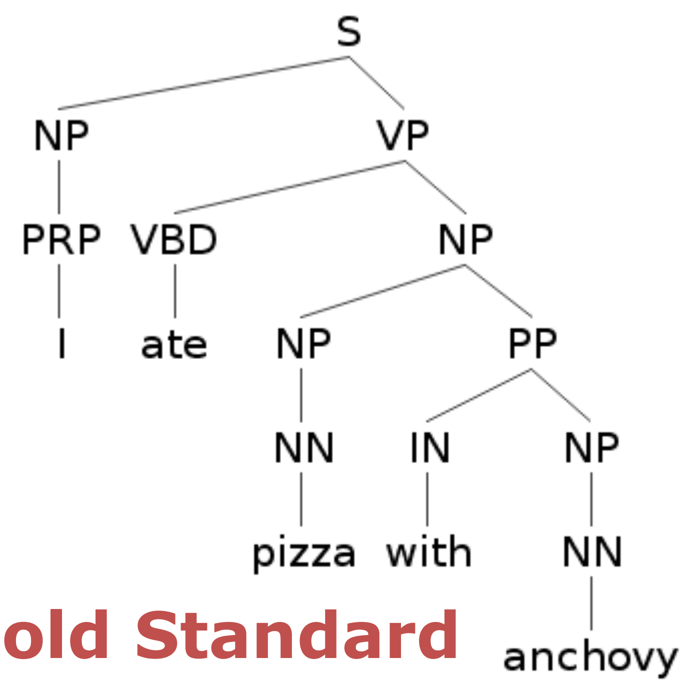
- F1-score = $\frac{(\beta^2+1)*Prec*Recall}{\beta^2*Precision + Recall}$

  Usually β= 1 (β>1 favor recall)

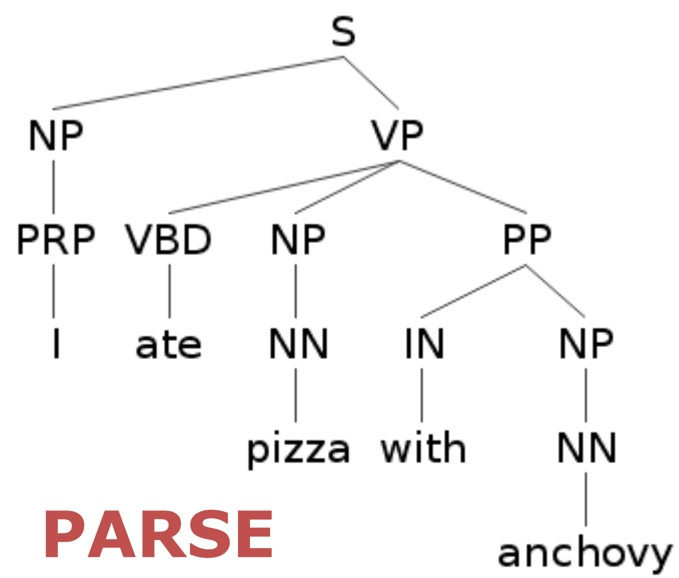
Search(Queen) = (2*0.5 *1) / (0.5 + 1) = 0.67

Search(Queen+Freddie) = (2*0.75*0.9) / (0.9+0.75) = 0.82

- For tree-structures

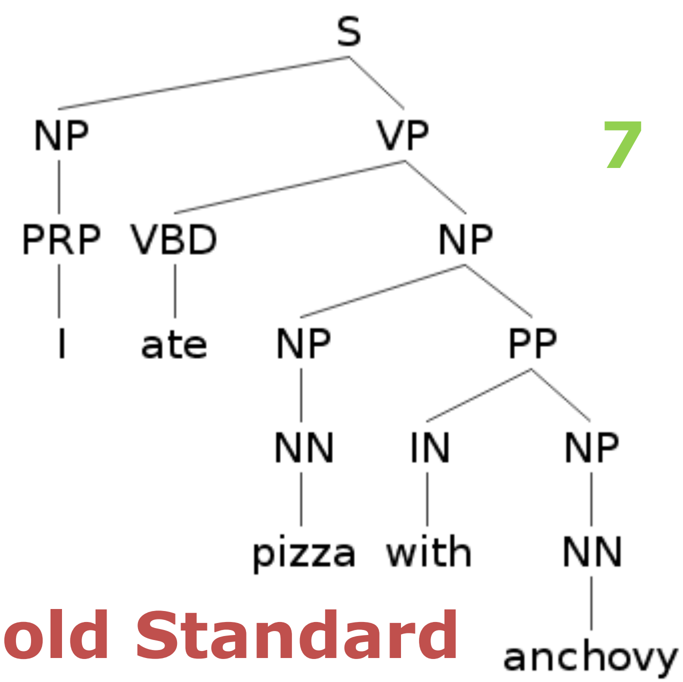


Gold Standard



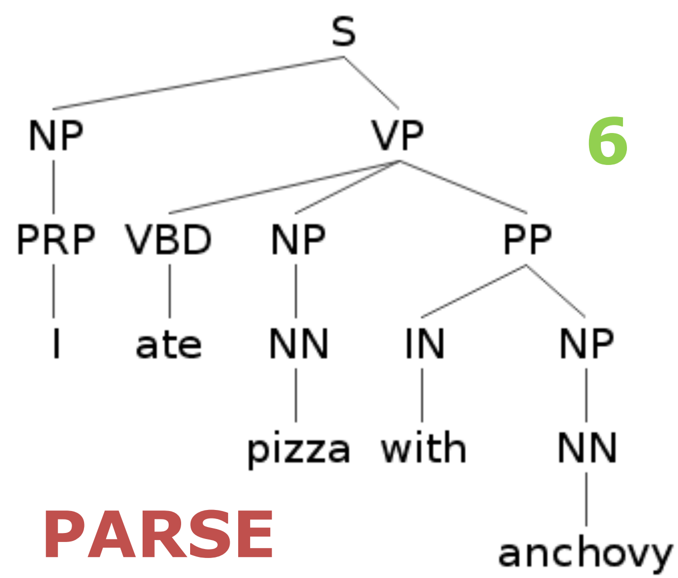
PARSE

- For tree-structures



7

Gold Standard



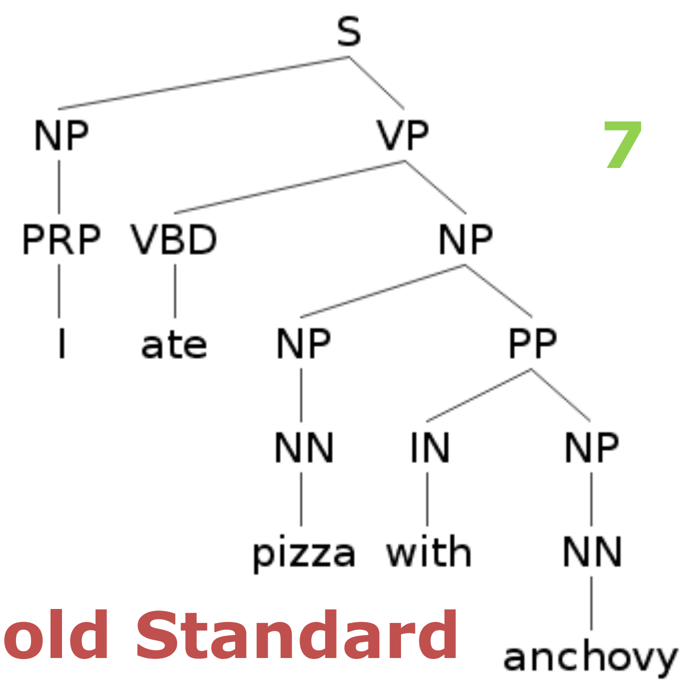
6

PARSE

- Count constituents

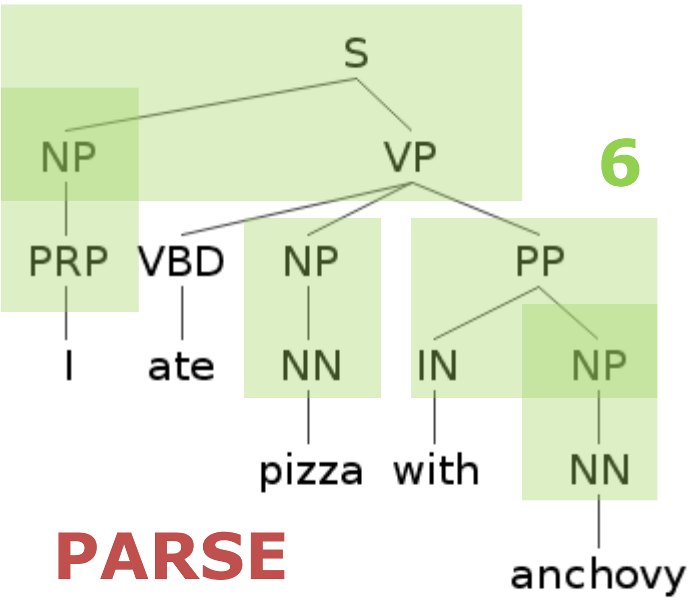
Precision/Recall

- For tree-structures



7

Gold Standard



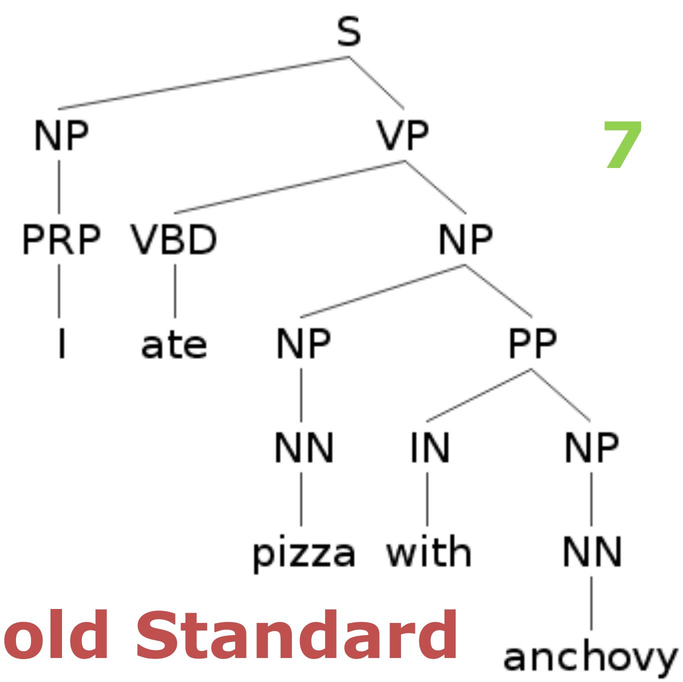
6

PARSE

- Count correctly found constituents: **5**

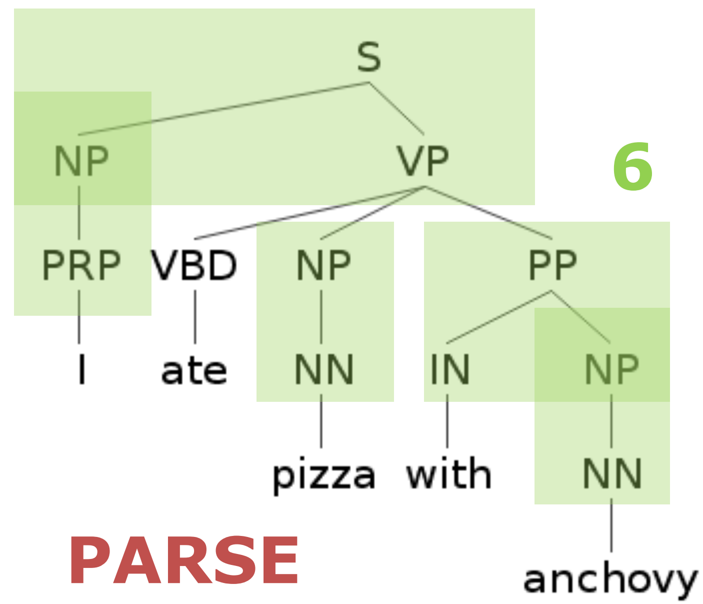
Precision/Recall

- For tree-structures



7

Gold Standard



6

PARSE

- Count correctly found constituents: **5**
- Precision = 5/6 = 0.83
- Recall = 5/7 = 0.71
- F-score = (2*0.83*0.71) / (0.83 + 0.71) = 0.77

Precision/Recall

- Is used as an evaluation metric for many NLP tasks
- Including word-sense disambiguation, named entity recognition, information retrieval, …

And… Python exercises on regular expressions for which the output is a set of words ☺

Assignment

```
>>> import nltk
>>> nltk.app.chartparser()
```

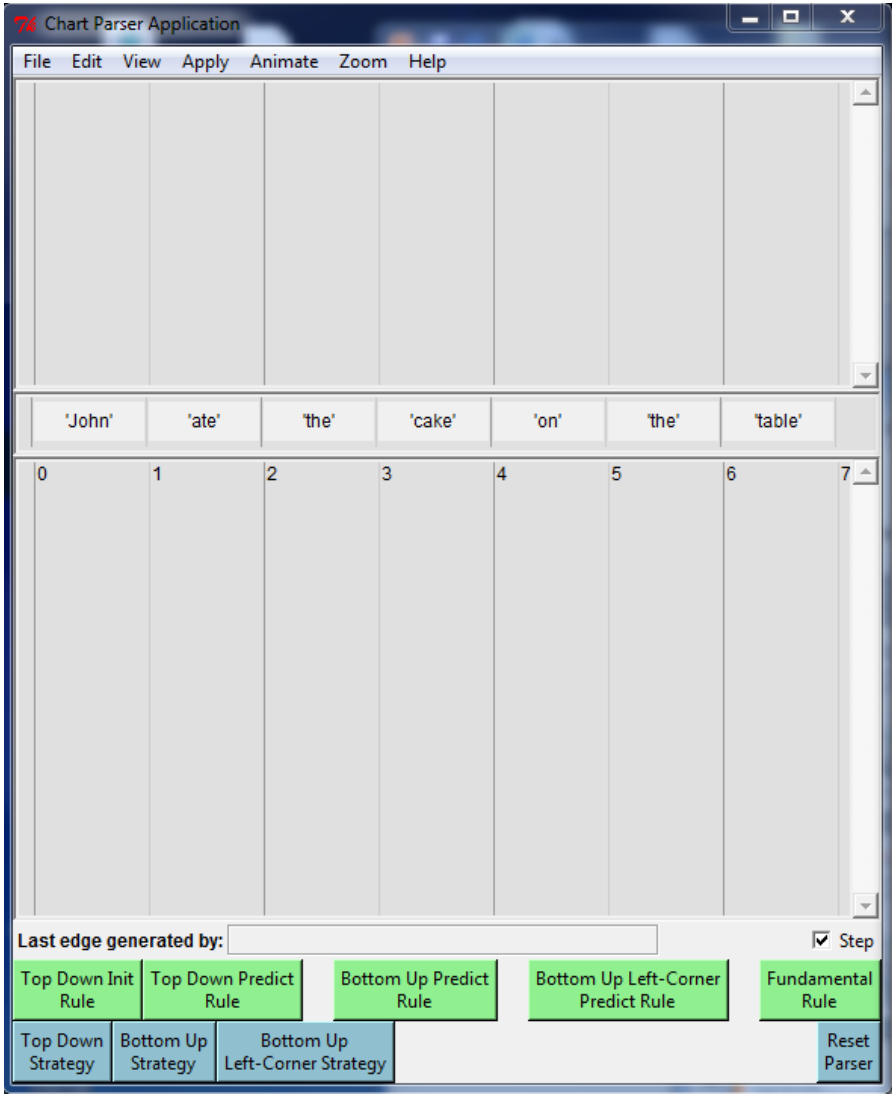
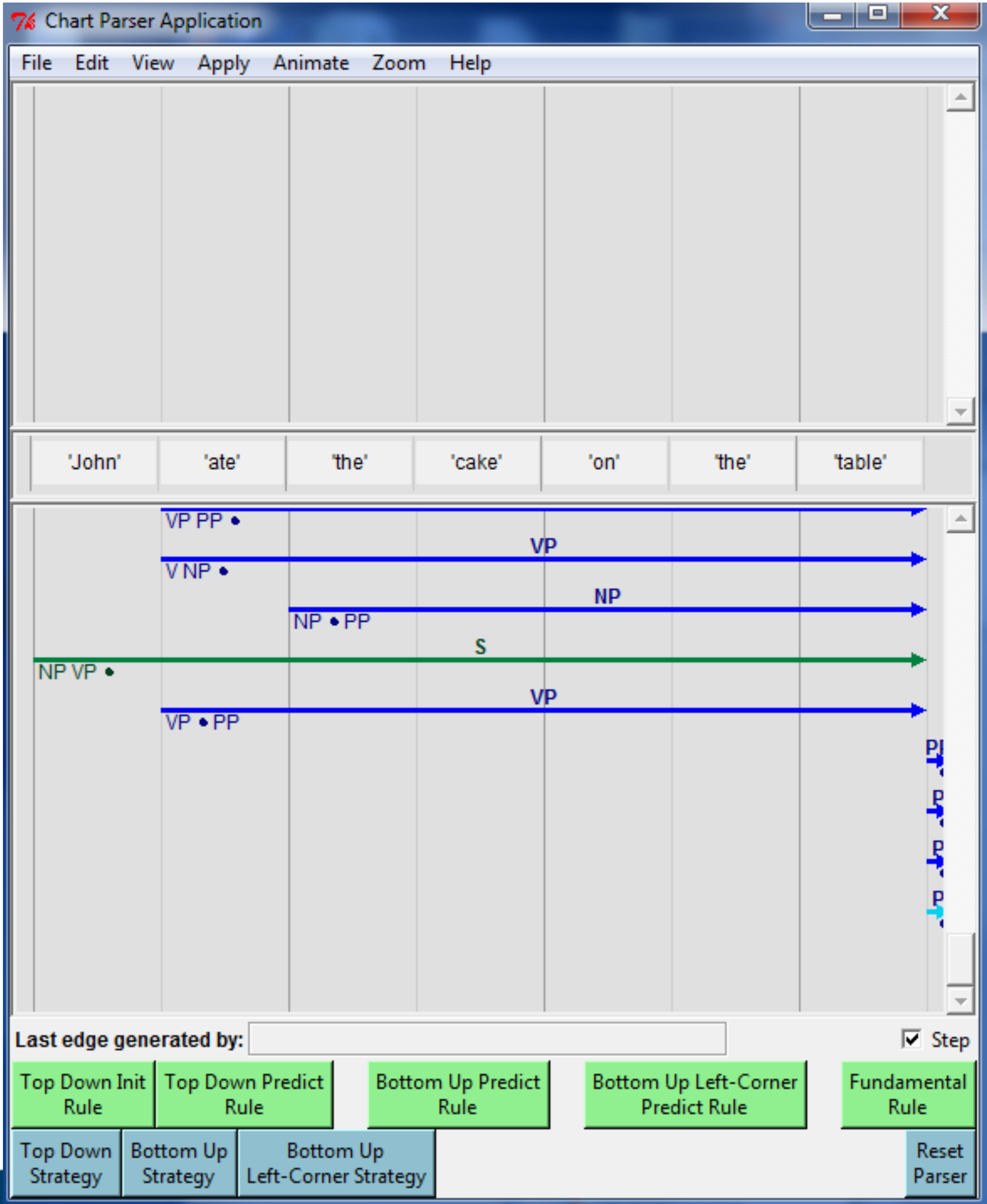


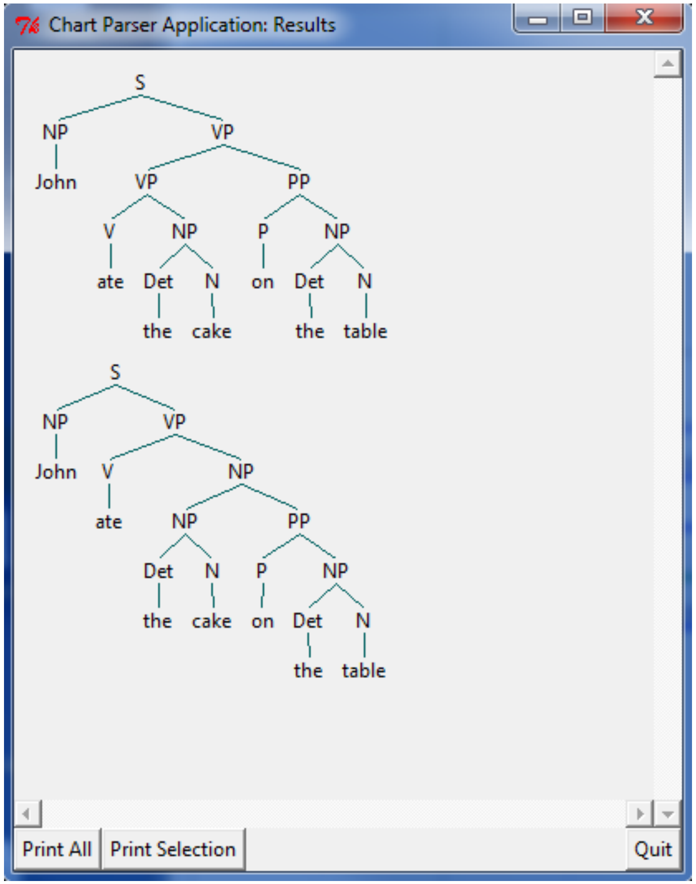
Chart Parser Application
File Edit View Apply Animate Zoom Help
'John' 'ate' 'the' 'cake' 'on' 'the' 'table'
0 1 2 3 4 5 6 7
Last edge generated by:
Step
Top Down Init Rule
Top Down Predict Rule
Bottom Up Predict Rule
Bottom Up Left-Corner Predict Rule
Fundamental Rule
Top Down Strategy
Bottom Up Strategy
Bottom Up Left-Corner Strategy
Reset Parser

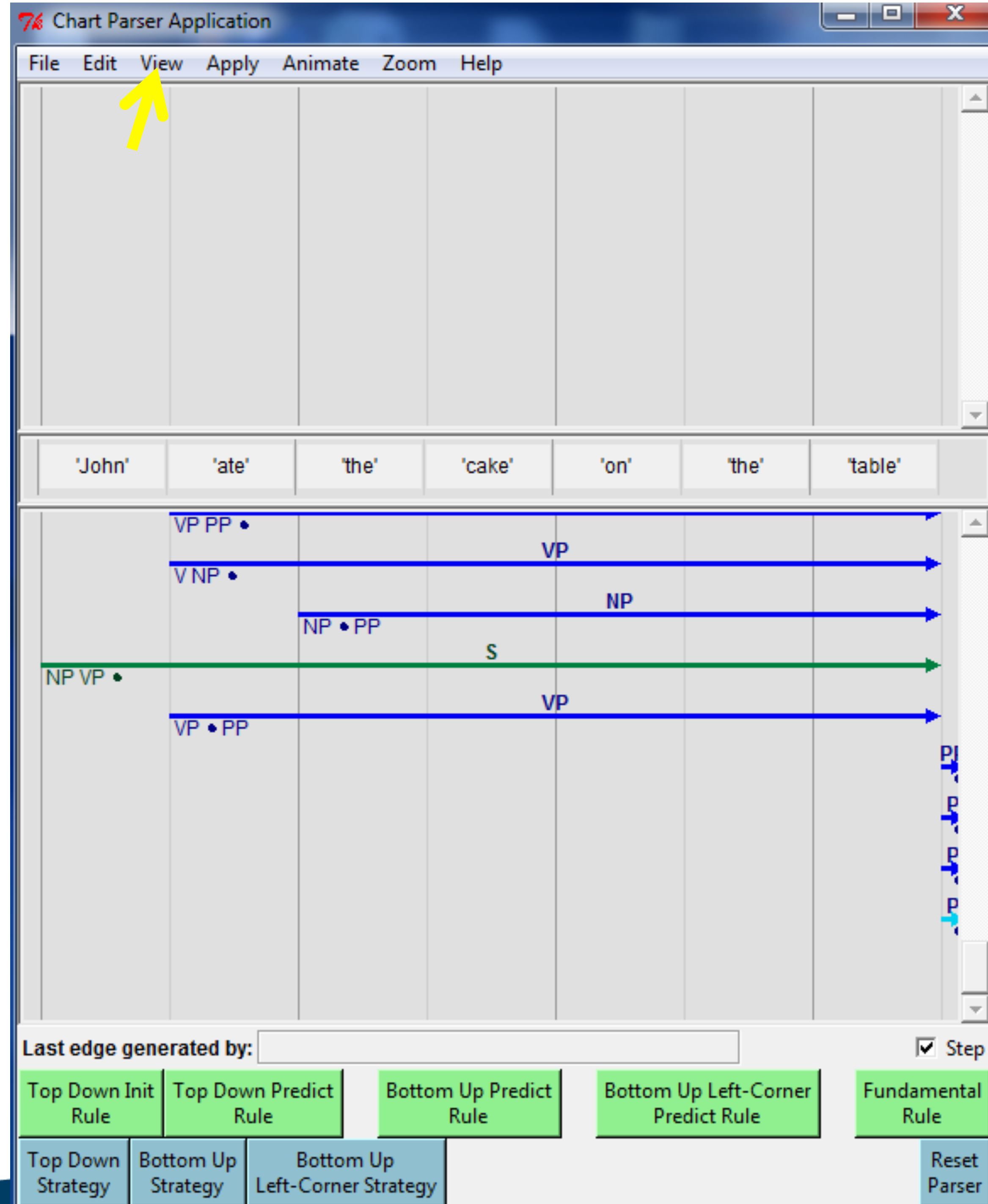
1. Keep clicking "top down strategy" or "bottom up strategy"
2. You should end up with a screen like →



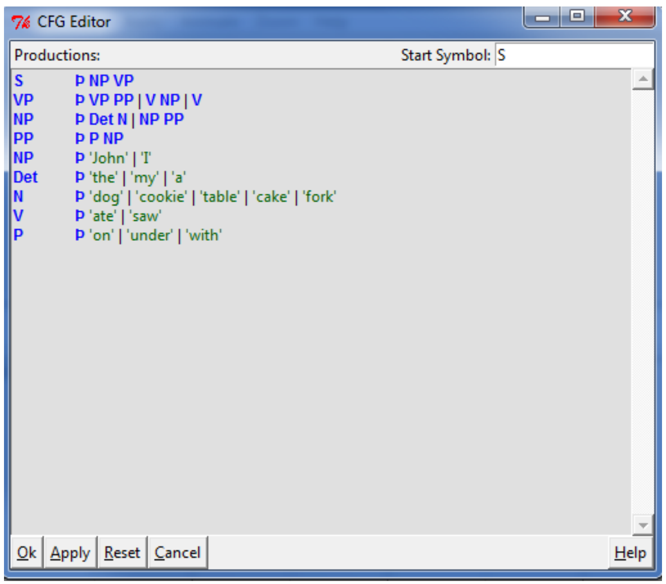
Universiteit Antwerpen

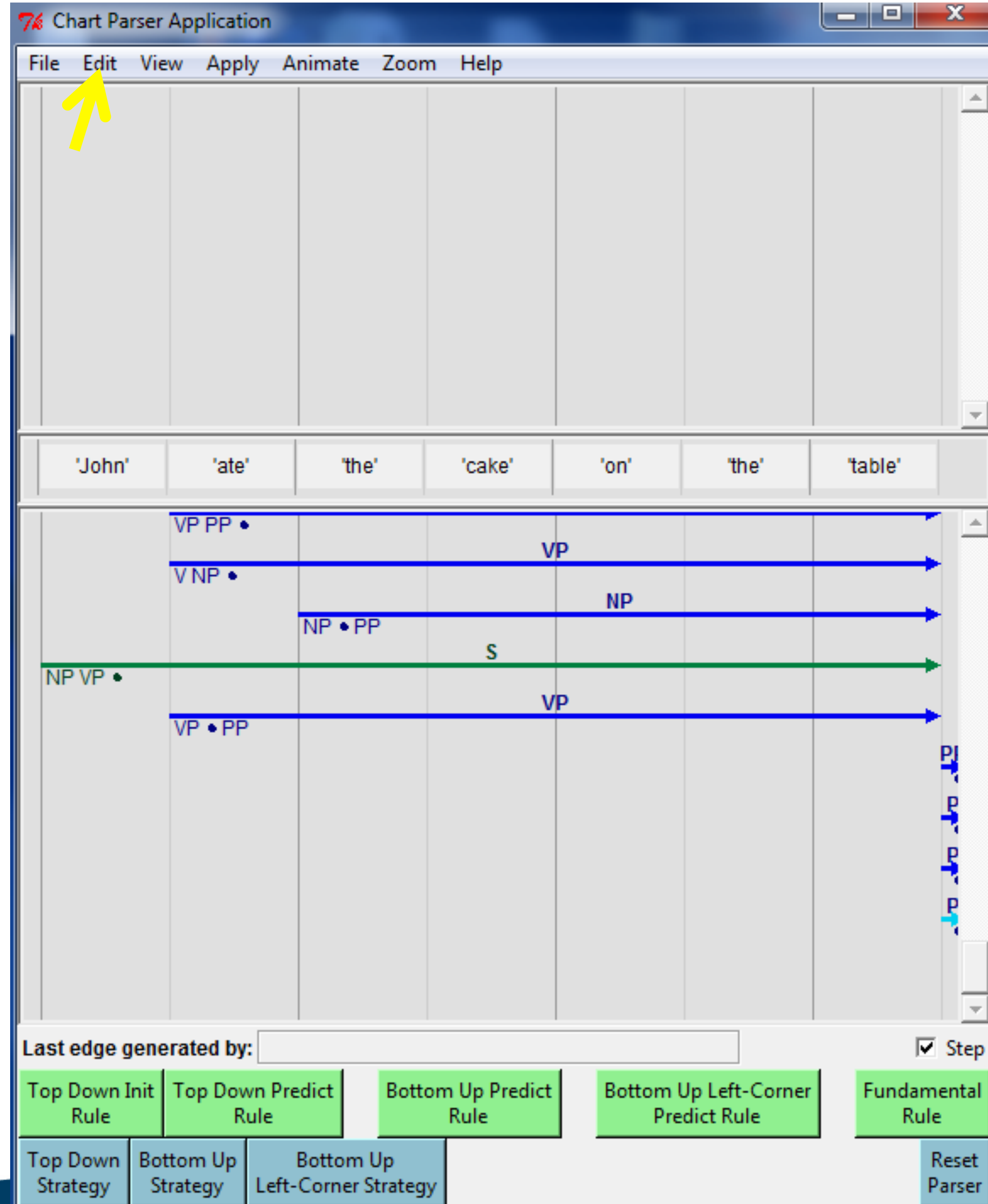
1. In the menu View>>Results you can see the generated parse trees





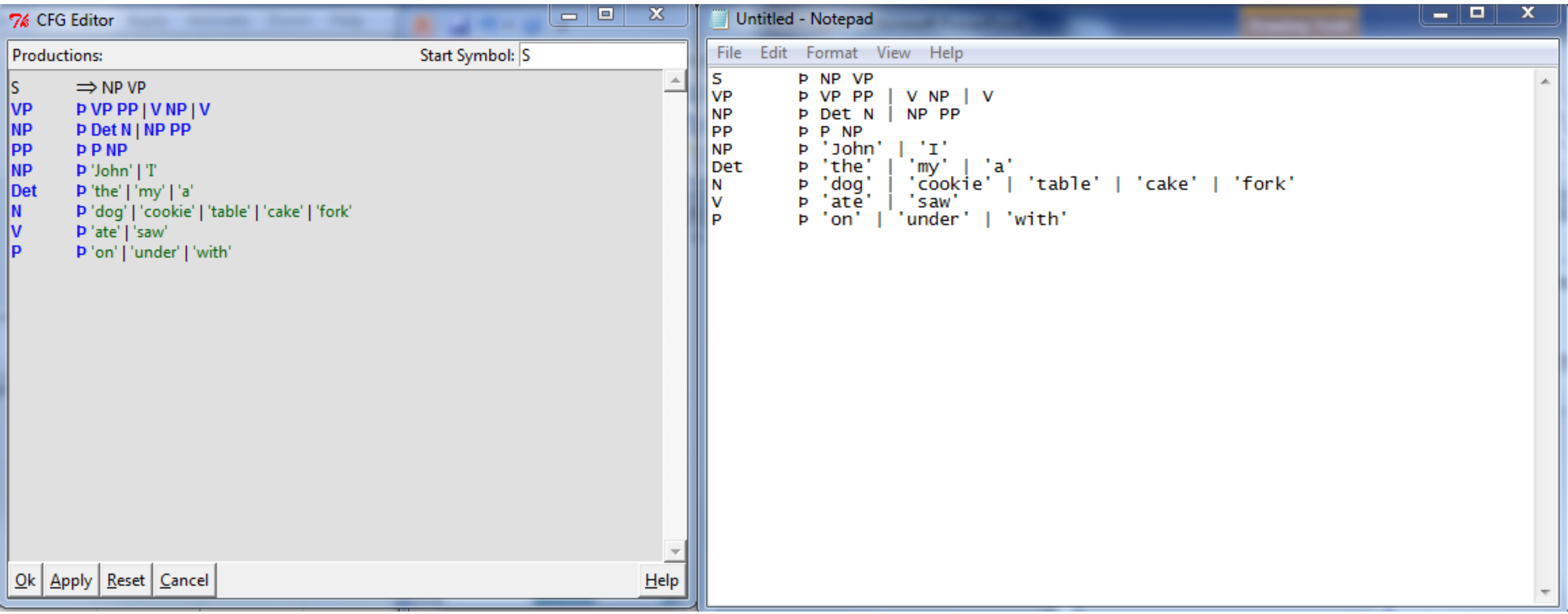
1. CTRL-G (or Edit>>Edit Grammar) will bring up the grammar editor





Unfortunately, on some operation systems, there is a bug in this editor. In this case, you cannot edit the grammar in this window directly.

1. Open a simple text-editor, such as Notepad on windows
2. Select the text in the grammar editor, copy it (CTRL+C) and paste it (CTRL+V) in the text editor

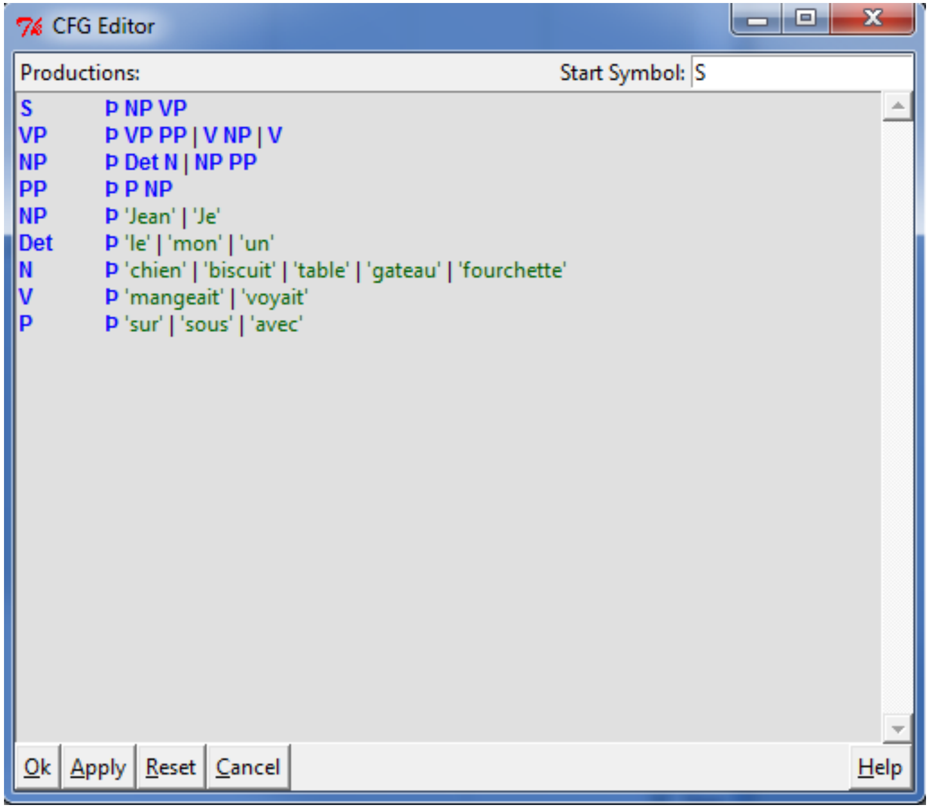


1. Write/edit your grammar in the text editor. In this example, I just translated the lexical items into French

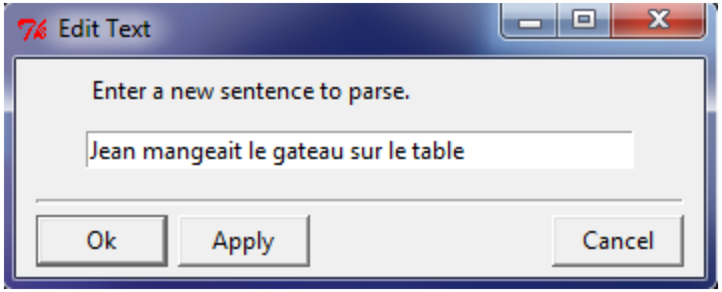
```
S       Þ NP VP
VP      Þ VP PP | V NP | V
NP      Þ Det N | NP PP
PP      Þ P NP
NP      Þ 'Jean' | 'Je'
Det     Þ 'le' | 'mon' | 'un'
N       Þ 'chien' | 'biscuit' | 'table' | 'gateau' | 'fourchette'
V       Þ 'mangeait' | 'voyait'
P       Þ 'sur' | 'sous' | 'avec'
```

Note: NLTK does not handle accents well. Therefore I wrote ‘gateau’ instead of ‘gâteau’.

1. When you’re done. Copy it and paste it back in the grammar editor and press ‘ok’.
2. (if you’re affected by the bug, don’t close the text editor. Keep editing your grammar in the text editor. If you re-open the grammar editor of NLTK again, the bug will generate bad characters in the grammar.



1. Then edit the sentence you want to parse with CTRL-T (or Edit>>Edit Text)



2. Repeat the steps from slides 108 and 109 to see your parse tree.

Note that this grammar overgenerates, e.g. *Je mangeait la fourchette
Find a short text (20 words) and write one (1!) grammar that can parse each sentence of that text.
The difficulty of the sentences, the coverage of your grammar, its overgeneration and its level of detail is up to you to decide

Send the text, grammar and a small report on how your parser handles the syntactic bottlenecks in your text. If there are multiple structures for 1 sentence, calculate precision/recall for the structures. Send your report to **guy.depauw@uantwerpen.be.**

Tip: turn your 20 word text into a treebank and induce a CFG from it
Deadline: Monday 15 December