Parsing

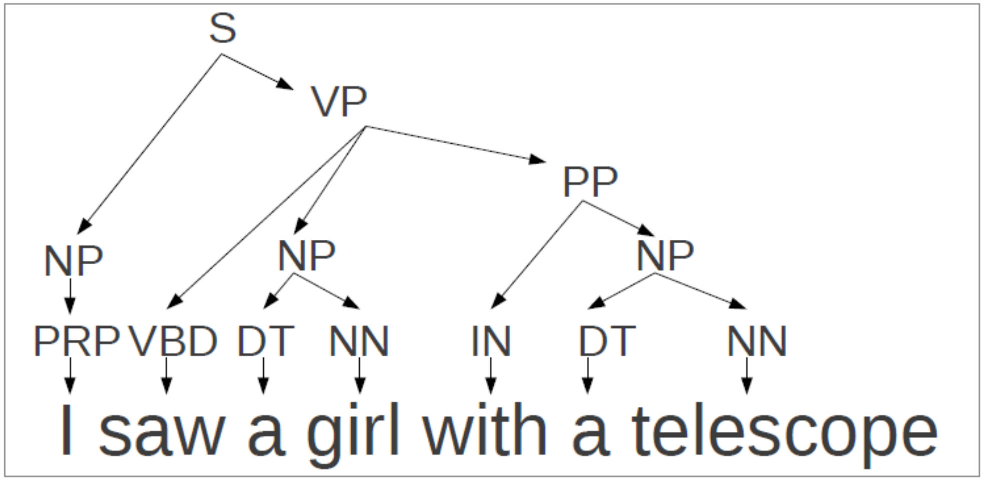
Natural Language Processing

Master in Business Analytics and Big Data

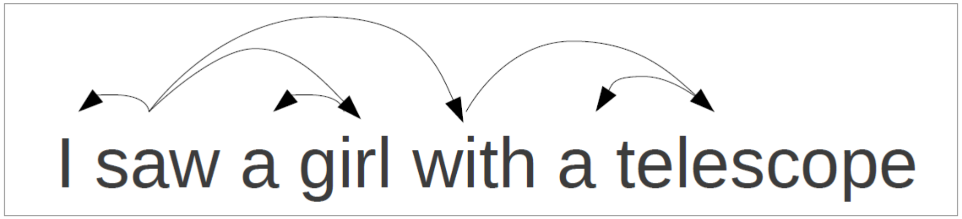
acastellanos@faculty.ie.edu

Two Types of Parsing

- **Constituency**: Phrases and their recursive structure



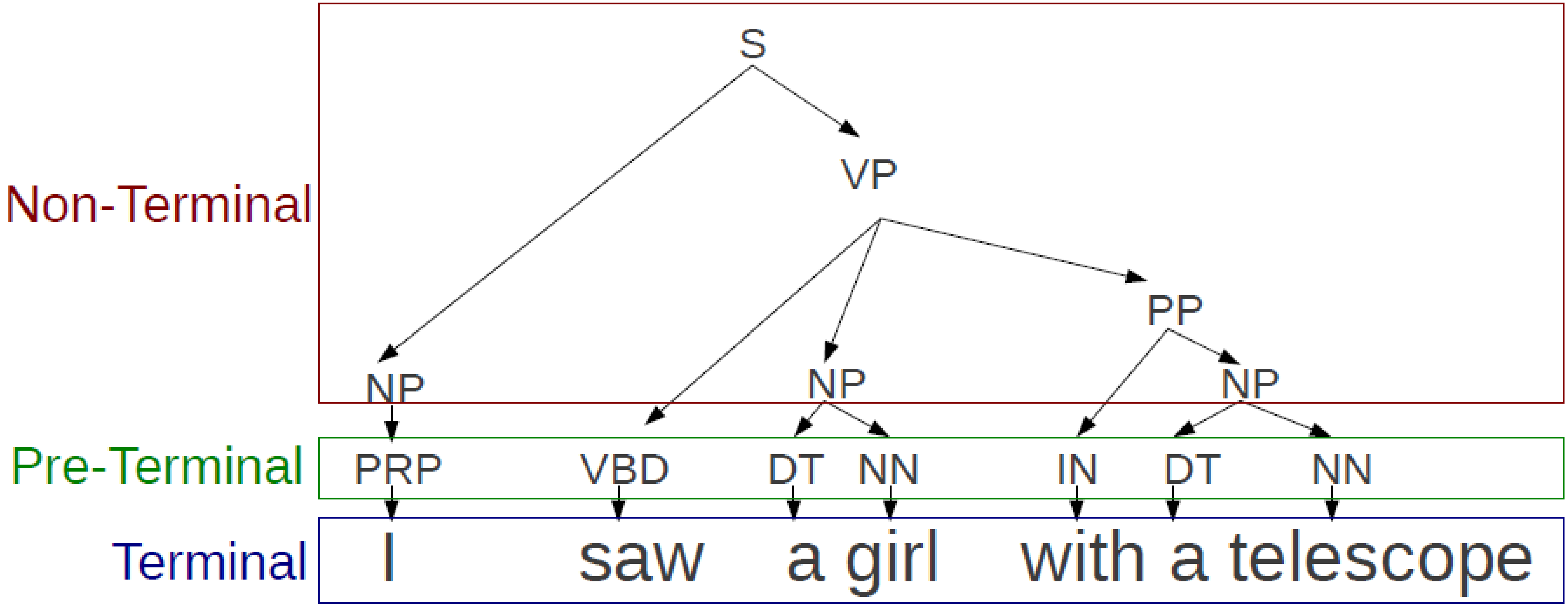
- **Dependency**: Relationships between words



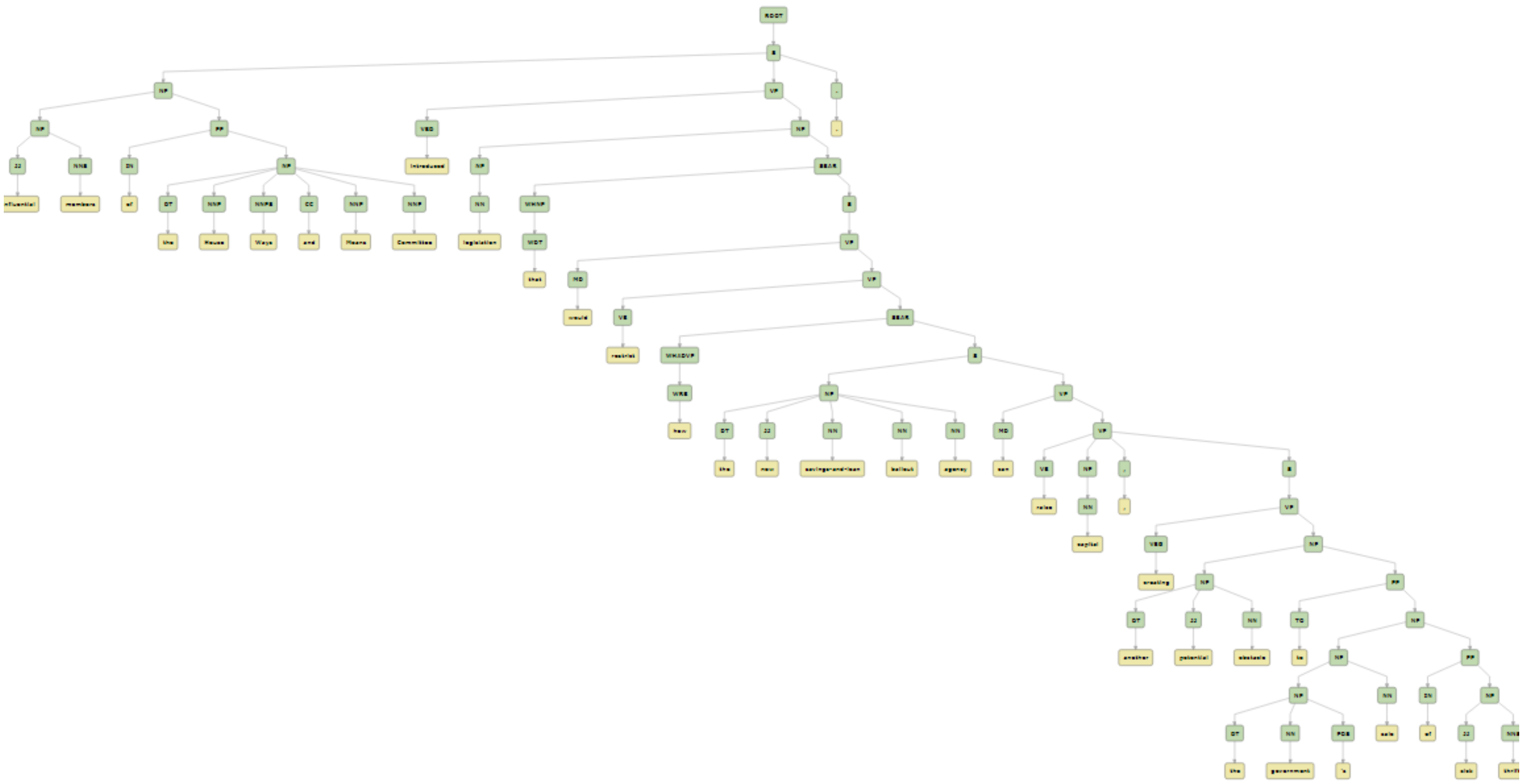
Constituency Parsing

- Words → nested constituents.
- What is a constituent?
  - **Distribution**: unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about
  - **Substitution/expansion**:
    - I sat [on the box/right on top of the box/there].
  - **Coordination**, regular internal structure, no intrusion, fragments, semantics, …

Constituency Parsing

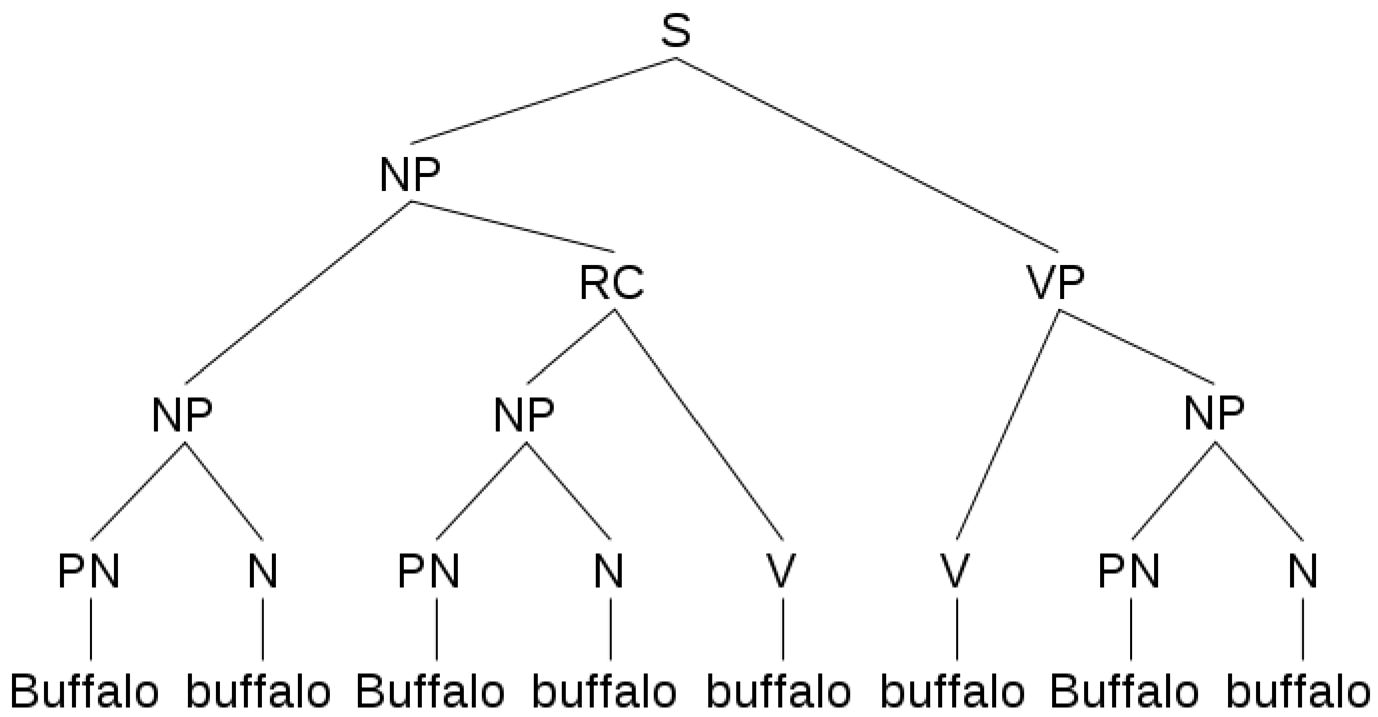


Constituency Parsing



Influential members of the House Ways and Means Committee introduced legislation that would restrict how the new savings-and-loan bailout agency can raise capital, creating another potential obstacle to the government's sale of sick thrifts.

Constituency Parsing



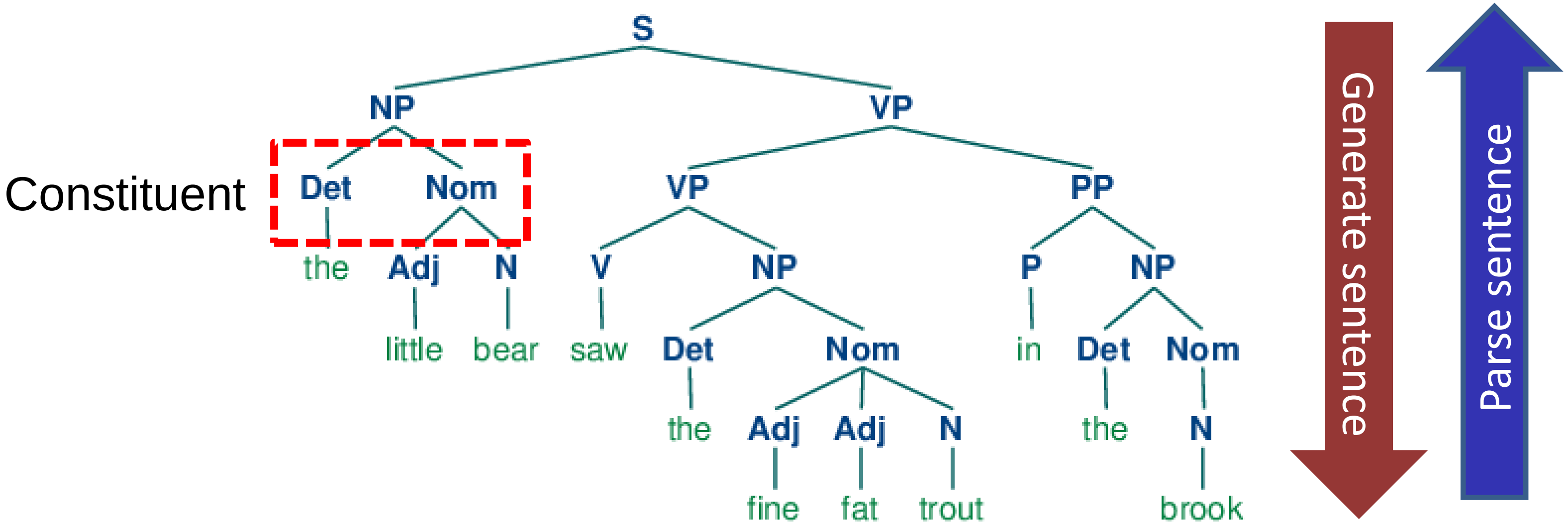
https://en.wikipedia.org/wiki/Buffalo_buffalo_Buffalo_buffalo_buffalo_buffalo_Buffalo_buffalo

Classical ("Pre 1990") NLP Parsing

- Wrote symbolic grammar (CFG) and lexicon

Grammar (CFG)		**Lexicon**
S $\rightarrow$ NP VP	VP $\rightarrow$ V NP	NN $\rightarrow$ *interest*
NP $\rightarrow$ (DT) NN	VP $\rightarrow$ VBP NP	NNS $\rightarrow$ *rates*
NP $\rightarrow$ NN NNS	VP $\rightarrow$ VBP NP PP	NNS $\rightarrow$ *raises*
NP $\rightarrow$ NP PP	PP $\rightarrow$ IN PP	VBP $\rightarrow$ *interest*
		VBZ $\rightarrow$ *rates*

Grammars


Constituent
S
NP
VP
Det
Nom
VP
PP
the
Adj
N
V
NP
P
NP
little
bear
saw
Det
Nom
in
Det
Nom
the
Adj
Adj
N
the
N
fine
fat
trout
brook
Generate sentence
Parse sentence

Parsers

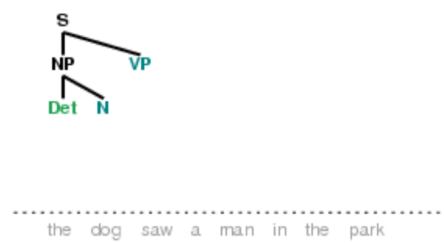
- **Build one or more constituent structure from a sentence**
  - Based on grammar productions
- **Top-down**
  - E.g. Recursive Descent Parsing
- **Bottom-up**
  - E.g. Shift-reduce parsing.

Recursive Descent Parsing

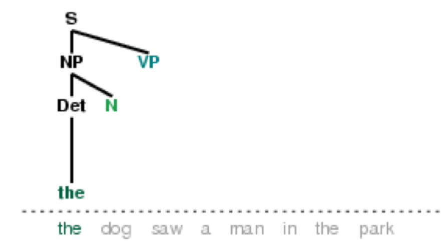
S -> NP VP
NP -> Det N PP
NP -> Det N
VP -> V NP PP
VP -> V NP
VP -> V
PP -> P NP
NP -> 'I'
Det -> 'the'
Det -> 'a'
N -> 'man'
N -> 'park'
N -> 'dog'
N -> 'telescope'
V -> 'ate'
V -> 'saw'
P -> 'in'
P -> 'under'
P -> 'with'

1. Initial stage

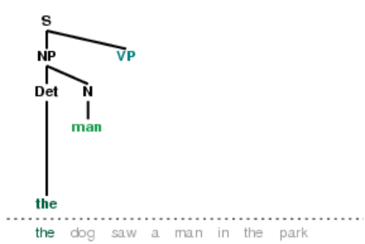
2. Second production



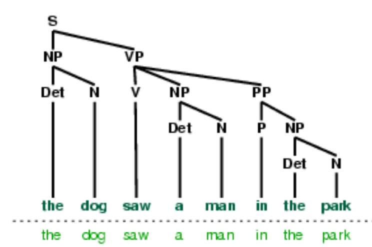
3. Matching *the*



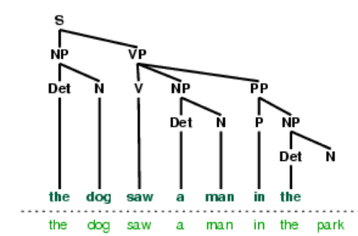
4. Cannot match *man*



5. Completed parse



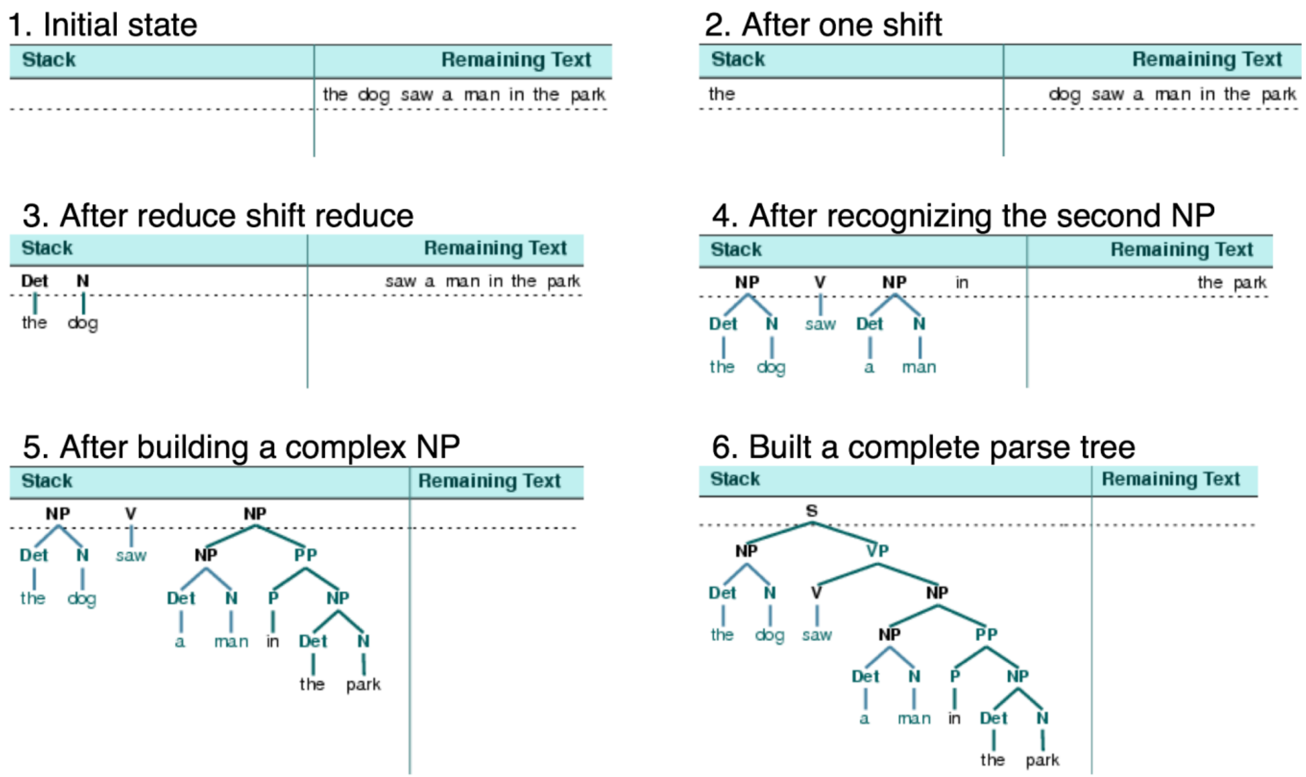
6. Backtracking



http://www.nltk.org/book/ch08.html

Shift-reduce parsing

S -> NP VP
NP -> Det N PP
NP -> Det N
VP -> V NP PP
VP -> V NP
VP -> V
PP -> P NP
NP -> 'I'
Det -> 'the'
Det -> 'a'
N -> 'man'
N -> 'park'
N -> 'dog'
N -> 'telescope'
V -> 'ate'
V -> 'saw'
P -> 'in'
P -> 'under'
P -> 'with'



http://www.nltk.org/book/ch08.html

Classical ("Pre 1990") NLP Parsing

- Wrote symbolic grammar (CFG) and lexicon

Grammar (CFG)		Lexicon
S $\rightarrow$ NP VP	VP $\rightarrow$ V NP	NN $\rightarrow$ *interest*
NP $\rightarrow$ (DT) NN	VP $\rightarrow$ VBP NP	NNS $\rightarrow$ *rates*
NP $\rightarrow$ NN NNS	VP $\rightarrow$ VBP NP PP	NNS $\rightarrow$ *raises*
NP $\rightarrow$ NP PP	PP $\rightarrow$ IN PP	VBP $\rightarrow$ *interest*
		VBZ $\rightarrow$ *rates*

- Deduct parses from words

Fed raises interest rates 0.5 percent

  - Minimal grammar: 36 parses
  - Simple 10 rule grammar: 592 parses
  - Real-size broad-coverage grammar: millions of parses

Classical ("Pre 1990") NLP Parsing

- Wrote symbolic grammar (CFG or often richer) and lexicon

S → NP VP	NN → *interest*
NP → (DT) NN	NNS → *rates*
NP → NN NNS	NNS → *raises*
NP → NNP	VBP → *interest*
VP → V NP	VBZ → *rates*

- Used deduction systems to prove parses from words

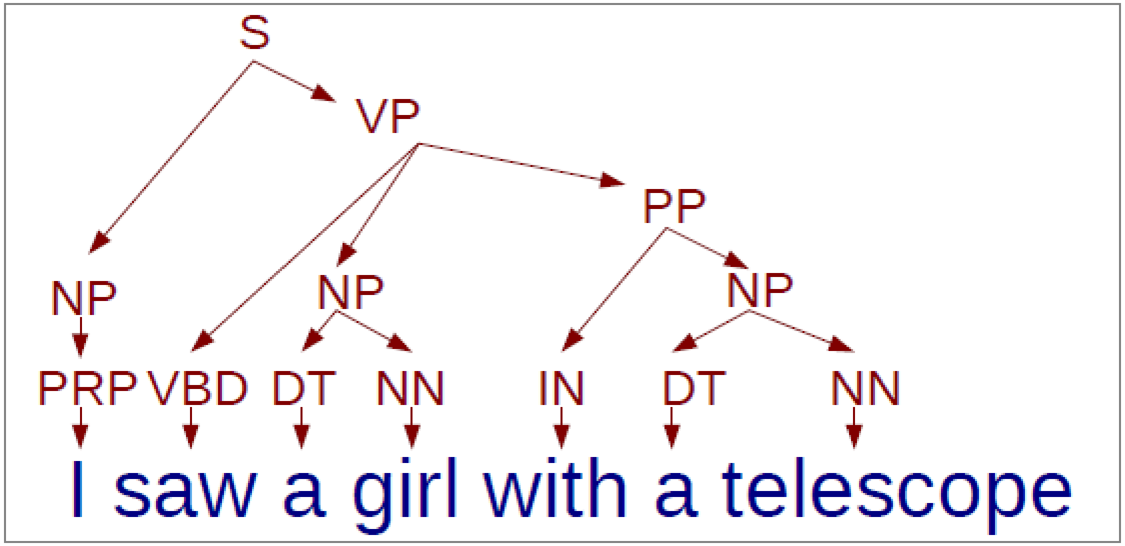
Fed raises interest rates 0.5 percent

- Minimal grammar: 36 parses
- Simple 10 rule grammar: 592 parses
- Real-size broad-coverage grammar: millions of parses

This scaled very badly and didn't give coverage.

Probabilistic Parsing

- Given a sentence X, predict its most probable parse tree T



$$\mathrm{argmax}_Y\, P(T|X)$$

Probabilistic Generative Parsing

- We assume some **probabilistic model** generated the sentence X and the parse tree T jointly

$$P(T, X)$$

- The parse tree with highest **joint probability** given X also has the highest **conditional probability.**

$$\text{argmax}_Y\, P(T|X) = \text{argmax}_Y\, P(T, X)$$

Probabilistic Generative Parsing

- **PCFG Learning**

$$P(A \rightarrow BC) = P(B, C|A) = \frac{count(A \rightarrow BC)}{count(A \rightarrow \backslash *)}$$

```
tbank_grammar = nltk.grammar.induce_pcfg(Nonterminal('S'), tbank_productions)
```

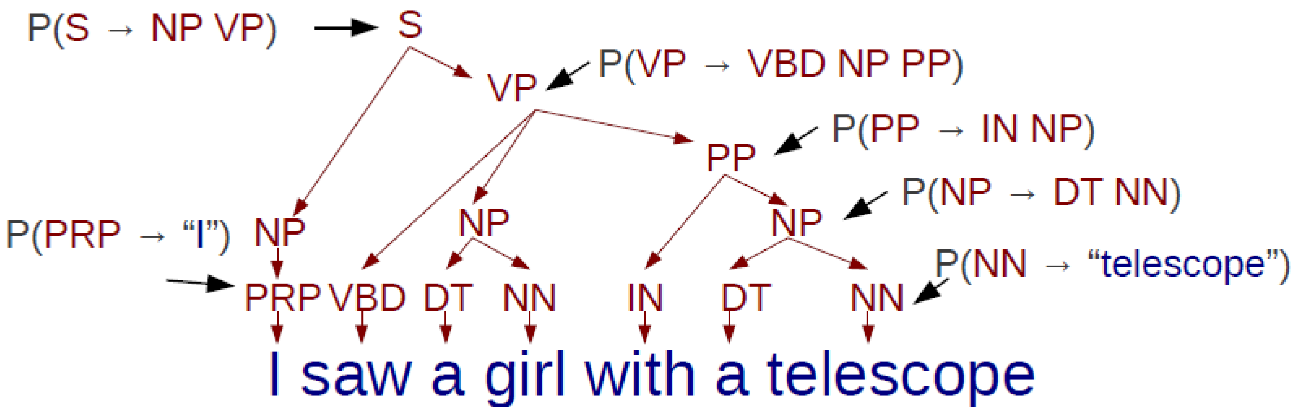
https://www.nltk.org/_modules/nltk/grammar.html#induce_pcfg

Probabilistic Context Free Grammar

- A **context-free grammar** is a tuple $< \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{S}, \boldsymbol{R} >$
  - N : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - T : the set of terminals (the words)
  - S : the start symbol
    - Often written as ROOT or TOP
  - R : the set of rules
    - Of the form $X \rightarrow Y1\ Y2\ \ldots\ Yk$, with $X, Yi \in N$
    - Examples: S → NP VP, VP → VP CC VP
- A **PCFG** adds:
  - A top-down production probability per rule $P(Y1\ Y2\ \ldots\ Yk \mid X)$

Probabilistic Context Free Grammar

- **PCFG**: Define probability for each node



- Parse tree probability is product of node probabilities

P(S → NP VP) * P(NP → PRP) * P(PRP → "I") * P(VP → VBD NP PP) *
P(VBD → "saw") * P(NP → DT NN) * P(DT → "a") * P(NN → "girl") * P(PP → IN NP) *
P(IN → "with") * P(NP → DT NN) * P(DT → "a") * P(NN → "telescope")

PCFG Example

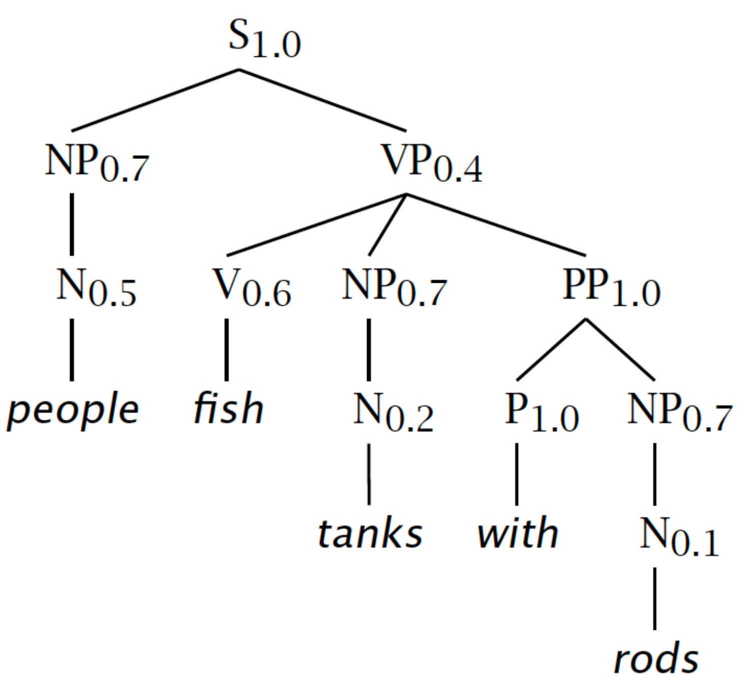
$S \rightarrow NP\ VP$	1.0
$VP \rightarrow V\ NP$	0.6
$VP \rightarrow V\ NP\ PP$	0.4
$NP \rightarrow NP\ NP$	0.1
$NP \rightarrow NP\ PP$	0.2
$NP \rightarrow N$	0.7
$PP \rightarrow P\ NP$	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

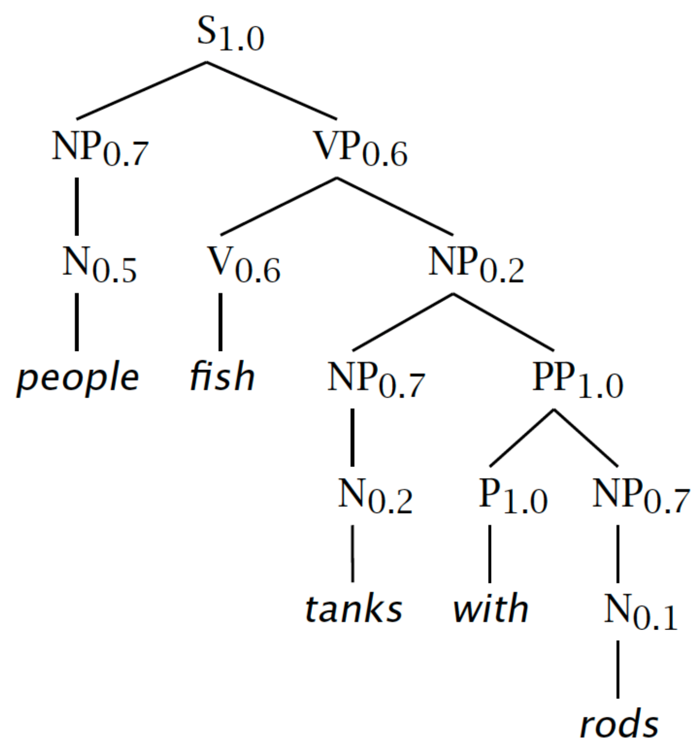
PCFG Example

S = `people fish tanks with rods`

t_1:

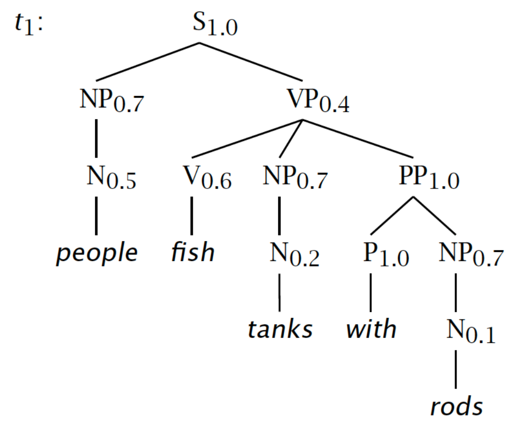


t_2:

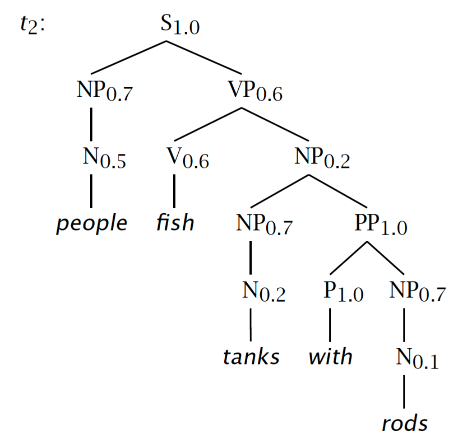


PCFG Example

- $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 1.0 \times 0.2 \times 0.7 \times 0.1 = \mathbf{0.0008232}$



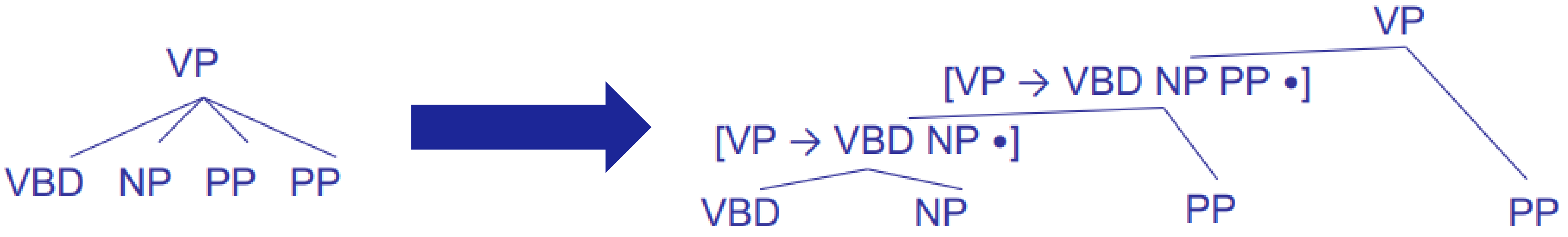
- $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = \mathbf{0.00024696}$



- **PCFG would choose t_1**

Chomsky Normal Form

- All rules are of the form $\boldsymbol{X \rightarrow Y\ Z}$ or $\boldsymbol{X \rightarrow w}$
  - $X, Y, Z \in N$ and $w \in T$
  - Empties and unaries are removed recursively
  - n-ary rules → new nonterminals (n > 2)
- Doesn't change the weak generative capacity of a CFG
  - That is, it recognizes the same language
- Makes parsing easier!



Chomsky Normal Form

S $\rightarrow$ NP VP

VP $\rightarrow$ V NP

VP $\rightarrow$ V NP PP

NP $\rightarrow$ NP NP

NP $\rightarrow$ NP PP

NP $\rightarrow$ N

NP $\rightarrow$ *e*

PP $\rightarrow$ P NP

N $\rightarrow$ *people*

N $\rightarrow$ *fish*

N $\rightarrow$ *tanks*

N $\rightarrow$ *rods*

V $\rightarrow$ *people*

V $\rightarrow$ *fish*

V $\rightarrow$ *tanks*

P $\rightarrow$ *with*

Chomsky Normal Form

Step 1: Remove epsilon rules

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ N
NP $\rightarrow$ *e*
PP $\rightarrow$ P NP

Recognizing the same language? For every rule with NP, create a unary rule

N $\rightarrow$ *people*
N $\rightarrow$ *fish*
N $\rightarrow$ *tanks*
N $\rightarrow$ *rods*
V $\rightarrow$ *people*
V $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
P $\rightarrow$ *with*

Chomsky Normal Form

Step 1: Remove epsilon rules

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Recognizing the same language? Work your way down to propagate

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Just added a unary rule! Need to apply until they are all gone

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Just added a unary rule! Need to apply until they are all gone

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S → NP VP
VP → V NP
S → V NP
VP → V
~~S → V~~
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

Just added a unary rule! Need to apply until they are all gone

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
S $\rightarrow$ V NP
~~VP $\rightarrow$ V~~
VP $\rightarrow$ V NP PP
S $\rightarrow$ V NP PP
VP $\rightarrow$ V PP
S $\rightarrow$ V PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ PP
NP $\rightarrow$ N
PP $\rightarrow$ P NP
PP $\rightarrow$ P

N $\rightarrow$ *people*
N $\rightarrow$ *fish*
N $\rightarrow$ *tanks*
N $\rightarrow$ *rods*
V $\rightarrow$ *people*
V $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
P $\rightarrow$ *with*

Chomsky Normal Form

Step 2: Remove unary rules

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
~~NP → NP~~
NP → NP PP
NP → PP
~~NP → N~~
PP → P NP
PP → P

Recognizing the same language? Yes!

Only place N appears
So can get rid of it altogether

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form

Step 2: Remove unary rules

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
S $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
S $\rightarrow$ V NP PP
VP $\rightarrow$ V PP
S $\rightarrow$ V PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP PP
~~NP $\rightarrow$ PP~~
PP $\rightarrow$ P NP
~~PP $\rightarrow$ P~~

NP $\rightarrow$ *people*
NP $\rightarrow$ *fish*
NP $\rightarrow$ *tanks*
NP $\rightarrow$ *rods*
V $\rightarrow$ *people*
S $\rightarrow$ *people*
VP $\rightarrow$ *people*
V $\rightarrow$ *fish*
S $\rightarrow$ *fish*
VP $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
S $\rightarrow$ *tanks*
VP $\rightarrow$ *tanks*
P $\rightarrow$ *with*
PP $\rightarrow$ *with*

Chomsky Normal Form

Step 3: Binarize

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
S $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
S $\rightarrow$ V NP PP
VP $\rightarrow$ V PP
S $\rightarrow$ V PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP PP
PP $\rightarrow$ P NP

NP $\rightarrow$ *people*
NP $\rightarrow$ *fish*
NP $\rightarrow$ *tanks*
NP $\rightarrow$ *rods*
V $\rightarrow$ *people*
S $\rightarrow$ *people*
VP $\rightarrow$ *people*
V $\rightarrow$ *fish*
S $\rightarrow$ *fish*
VP $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
S $\rightarrow$ *tanks*
VP $\rightarrow$ *tanks*
P $\rightarrow$ *with*
PP $\rightarrow$ *with*

Chomsky Normal Form

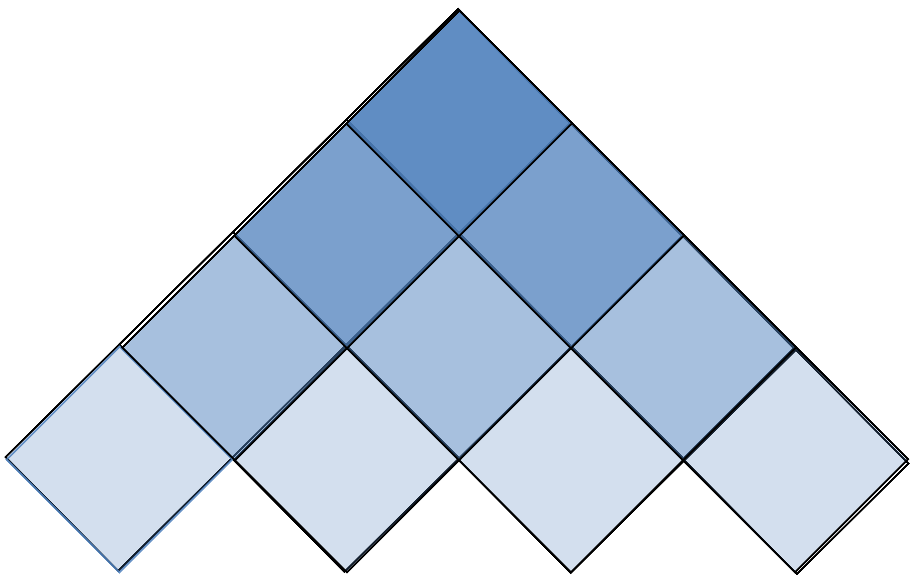
Step 3: Binarize

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
S $\rightarrow$ V NP
VP $\rightarrow$ V @VP_V
@VP_V $\rightarrow$ NP PP
S $\rightarrow$ V @S_V
@S_V $\rightarrow$ NP PP
VP $\rightarrow$ V PP
S $\rightarrow$ V PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP PP
PP $\rightarrow$ P NP

NP $\rightarrow$ *people*
NP $\rightarrow$ *fish*
NP $\rightarrow$ *tanks*
NP $\rightarrow$ *rods*
V $\rightarrow$ *people*
S $\rightarrow$ *people*
VP $\rightarrow$ *people*
V $\rightarrow$ *fish*
S $\rightarrow$ *fish*
VP $\rightarrow$ *fish*
V $\rightarrow$ *tanks*
S $\rightarrow$ *tanks*
VP $\rightarrow$ *tanks*
P $\rightarrow$ *with*
PP $\rightarrow$ *with*

CKY Algorithm

- Cocke-Kasami-Younger

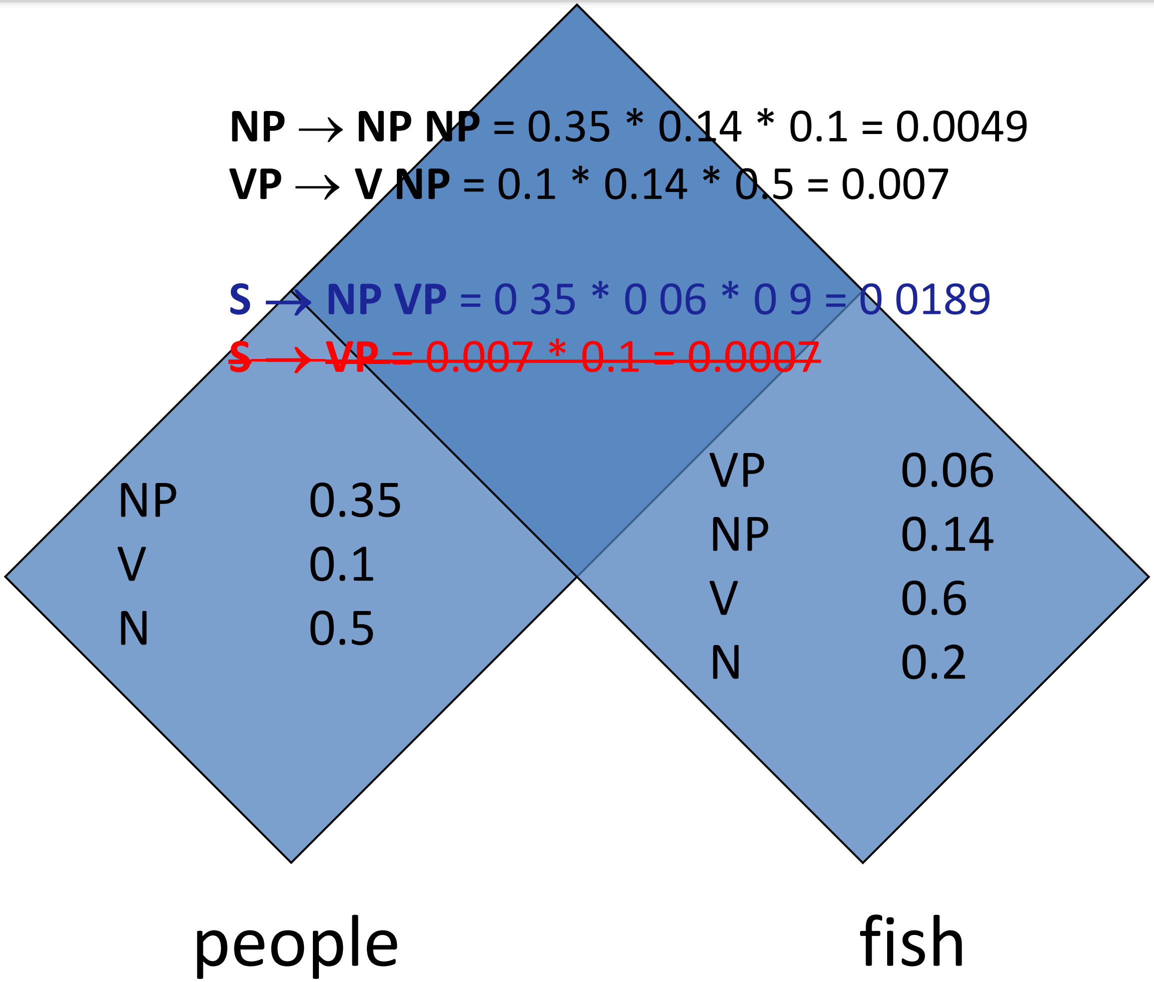


fish people fish tanks

CKY Algorithm

NP → NP NP = 0.35 * 0.14 * 0.1 = 0.0049
VP → V NP = 0.1 * 0.14 * 0.5 = 0.007

S → NP VP = 0 35 * 0 06 * 0 9 = 0 0189
~~S → VP = 0.007 * 0.1 = 0.0007~~



PCFG

$\boxed{S \rightarrow NP\ VP}$	0.9
$\boxed{S \rightarrow VP}$	0.1
$\boxed{VP \rightarrow V\ NP}$	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
$\boxed{NP \rightarrow NP\ NP}$	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

CKY Parsing: A Worked Example

$S \rightarrow NP\ VP$	0.9
$S \rightarrow VP$	0.1
$VP \rightarrow V\ NP$	0.5
$VP \rightarrow V$	0.1
$VP \rightarrow V\ @VP\_V$	0.3
$VP \rightarrow V\ PP$	0.1
$@VP\_V \rightarrow NP\ PP$	1.0
$NP \rightarrow NP\ NP$	0.1
$NP \rightarrow NP\ PP$	0.2
$NP \rightarrow N$	0.7
$PP \rightarrow P\ NP$	1.0

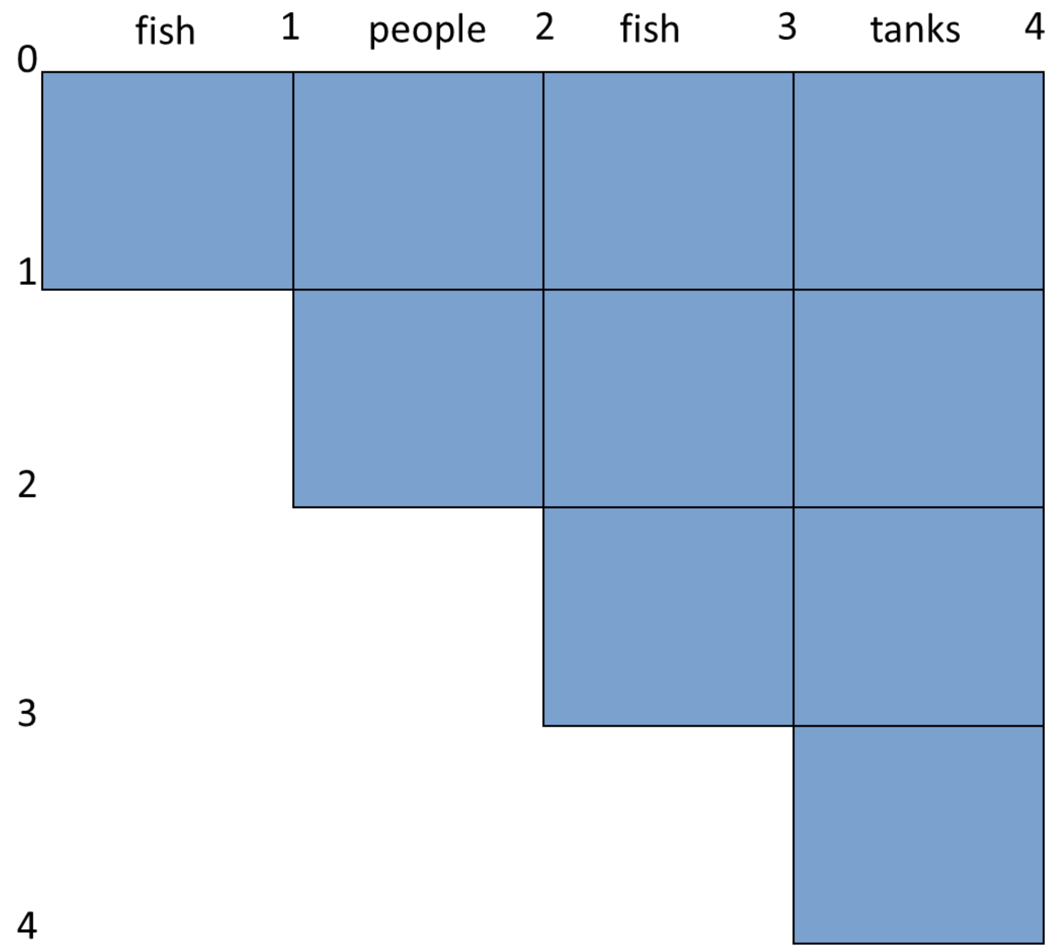
$N \rightarrow$ *people*	0.5
$N \rightarrow$ *fish*	0.2
$N \rightarrow$ *tanks*	0.2
$N \rightarrow$ *rods*	0.1
$V \rightarrow$ *people*	0.1
$V \rightarrow$ *fish*	0.6
$V \rightarrow$ *tanks*	0.3
$P \rightarrow$ *with*	1.0

CKY Parsing: A Worked Example

	fish 1	people 2	fish 3	tanks 4
0	score[0][1]	score[0][2]	score[0][3]	score[0][4]
1		score[1][2]	score[1][3]	score[1][4]
2			score[2][3]	score[2][4]
3				score[3][4]
4				

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**
N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**



CKY Parsing: A Worked Example

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0

N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6			
1		N → people 0.5 V → people 0.1		
2			N → fish 0.2 V → fish 0.6	
3				N → tanks 0.2 V → tanks 0.1

```
for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i]);
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**

N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 **NP → N 0.14** **VP → V 0.06** **S → VP 0.006**			
1		N → people 0.5 V → people 0.1 **NP → N 0.35** **VP → V 0.01** **S → VP 0.001**		
2			N → fish 0.2 V → fish 0.6 **NP → N 0.14** **VP → V 0.06** **S → VP 0.006**	
3				N → tanks 0.2 V → tanks 0.1 **NP → N 0.14** **VP → V 0.03** **S → VP 0.003**

```
// handle unaries
boolean added = true
  while added
    added = false
    for A, B in nonterms
      if score[i][i+1][B] > 0 && A->B in grammar
        prob = P(A->B)*score[i][i+1][B]
        if(prob > score[i][i+1][A])
          score[i][i+1][A] = prob
          back[i][i+1][A] = B
          added = true
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**
N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	fish (0–1)	people (1–2)	fish (2–3)	tanks (3–4)
0	N → fish 0.2 V → fish 0.6 **NP → N 0.14** VP → V 0.06 S → VP 0.006	**NP → NP NP 0.0049** **VP → V NP 0.105** **S → NP VP 0.00126**		
1		N → people 0.5 V → people 0.1 **NP → N 0.35** VP → V 0.01 S → VP 0.001	**NP → NPNP 0.0049** **VP → V NP 0.007** **S → NP VP 0.0189**	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	**NP → NPNP 0.00196** **VP → V NP 0.042** **S → NP VP 0.00378**
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

```
prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
if (prob > score[begin][end][A])
    score[begin]end][A] = prob
    back[begin][end][A] = new Triple(split,B,C)
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP 0.1	
NP → NP PP 0.2	
NP → N	**0.7**
PP → P NP	**1.0**
N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	fish (0–1)	people (1–2)	fish (2–3)	tanks (3–4)
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NPNP 0.0049 **VP → VNP 0.105** S → VP 0.0105		
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NPNP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NPNP 0.00196 **VP → V NP 0.042** S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

```
//handle unaries
boolean added = true
while added
  added = false
  for A, B in nonterms
    prob = P(A->B)*score[begin][end][B];
    if prob > score[begin][end][A]
      score[begin][end][A] = prob
      back[begin][end][A] = B
      added = true
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**
N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	0 fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	**NP → NPNP 0.0000686** **VP → V NP 0.00147** **S → NP VP 0.000882**	
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NPNP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NPNP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

```
for split = begin+1 to end-1
   for A,B,C in nonterms
      prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
      if prob > score[begin][end][A]
         score[begin]end][A] = prob
         back[begin][end][A] = new Triple(split,B,C)
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**

N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	0 fish 1	1 people 2	2 fish 3	3 tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP→NPNP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NPNP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	**NP→NPNP 0.0000686** **VP → V NP 0.000098** **S → NP VP 0.01323**
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NPNP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

```
for split = begin+1 to end-1
   for A,B,C in nonterms
      prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
      if prob > score[begin][end][A]
         score[begin]end][A] = prob
         back[begin][end][A] = new Triple(split,B,C)
```

CKY Parsing: A Worked Example

S → NP VP	**0.9**
S → VP	**0.1**
VP → V NP	**0.5**
VP → V	**0.1**
VP → V @VP_V	**0.3**
VP → V PP	**0.1**
@VP_V → NP PP	**1.0**
NP → NP NP	**0.1**
NP → NP PP	**0.2**
NP → N	**0.7**
PP → P NP	**1.0**

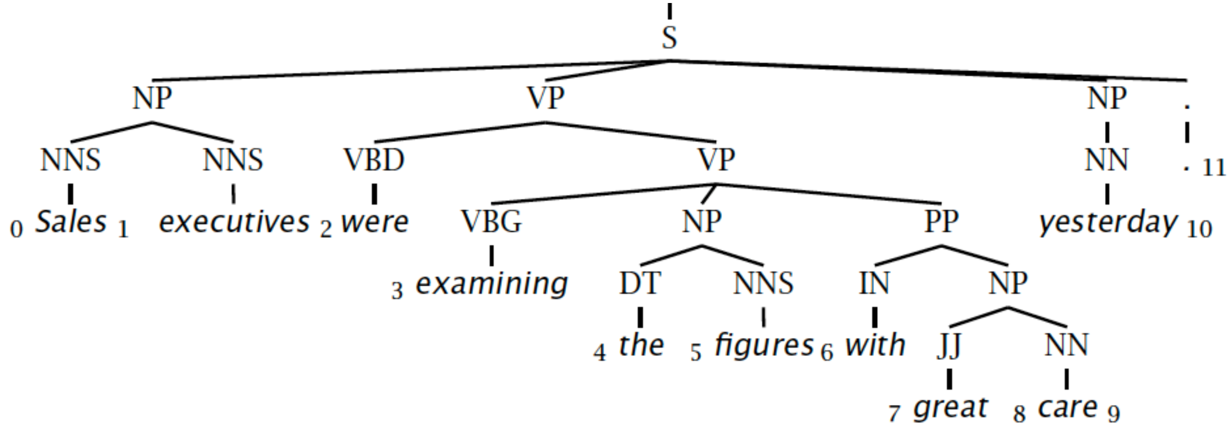
N → *people*	**0.5**
N → *fish*	**0.2**
N → *tanks*	**0.2**
N → *rods*	**0.1**
V → *people*	**0.1**
V → *fish*	**0.6**
V → *tanks*	**0.3**
P → *with*	**1.0**

	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP→NPNP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	**NP → NP NP 0.0000009604** **VP → V NP 0.00002058** **S → NP VP 0.00018522**
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NPNP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP→NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NPNP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003

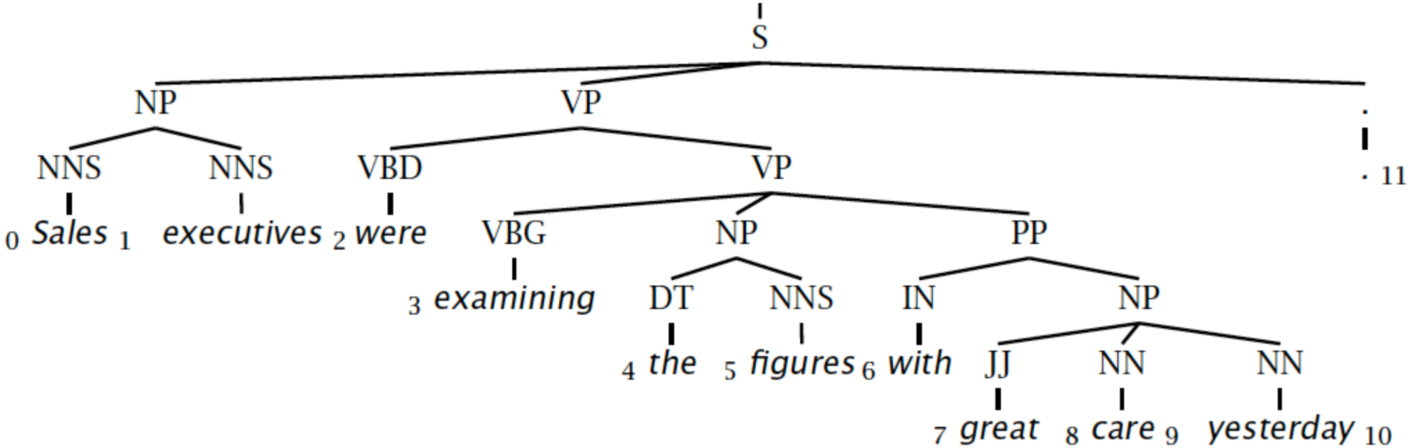
```
for split = begin+1 to end-1
   for A,B,C in nonterms
      prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
      if prob > score[begin][end][A]
         score[begin]end][A] = prob
         back[begin][end][A] = new Triple(split,B,C)
```

Evaluating constituency parsing

Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)



Evaluating constituency parsing

Gold standard brackets:

S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:

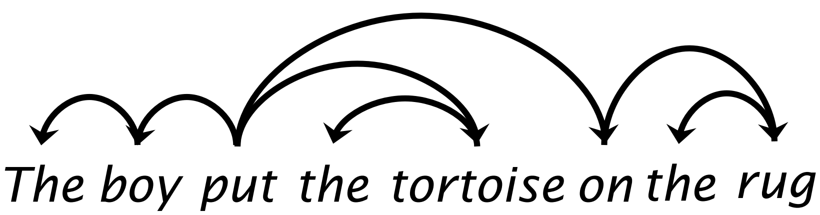
S-(0:11), **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

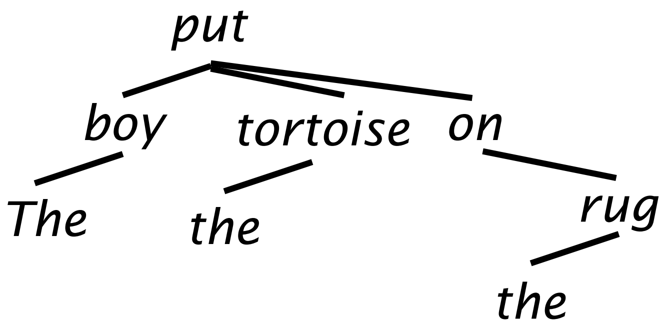
Labeled Precision	**3/7 = 42.9%**
Labeled Recall	**3/8 = 37.5%**
LP/LR F1	**40.0%**
Tagging Accuracy	**11/11 = 100.0%**

Penn WSJ parsing accuracy: about **73% LP/LR F1**

Dependency Parsing

- **Dependency structure**
  - Which **words depend on** (modify or are arguments of) **which other words.**
  - Defined as **directed graph** $\boldsymbol{G = (V, E)}$
    - **Connected:** For every node $i \in V$ there is a node $j \in V$, such that $i \rightarrow j \; or \; j \rightarrow i$
    - **Acyclic:** No cycles
    - **Single-head:** Have only one parent (except for root token)
  - Captures the **syntactic relations**

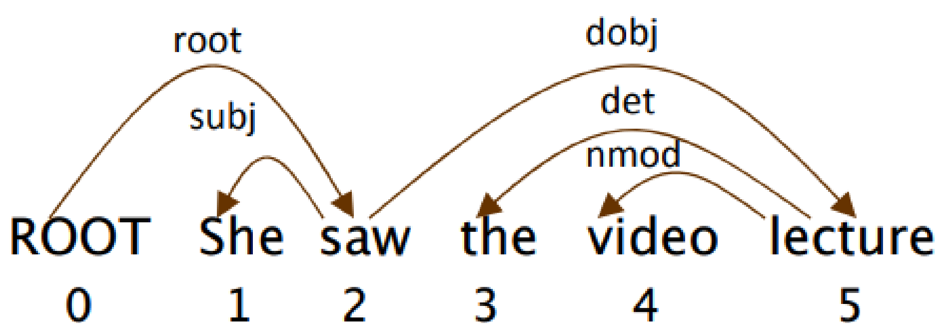




Methods of Dependency Parsing

- **Dynamic programming** (like in the CKY algorithm)
- **Graph** algorithms
- **Constraint Satisfaction**
  - Edges are eliminated that don't satisfy hard constraints
- **"Deterministic parsing"**
  - Greedy choice of attachments guided by machine learning classifiers

Evaluation



Unlabeled Attachment Score (UAS)
Labeled Attachment Score (LAS)
Label Accuracy (LA)

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%
LA = 3 / 5 = 60%

Gold

1	She	2	subj
2	saw	0	root
3	the	5	det
4	video	5	nmod
5	lecture	2	dobj

Parsed

1	She	2	subj
2	saw	0	root
3	the	4	det
4	video	5	vmod
5	lecture	2	iobj

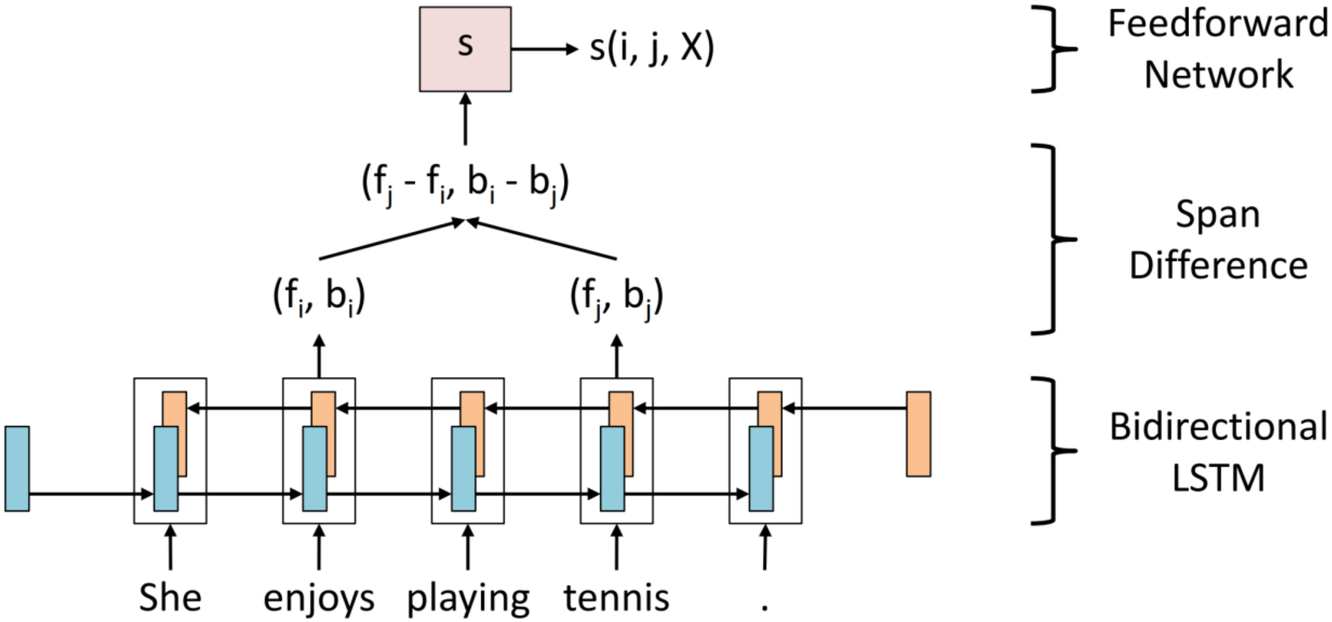
Evaluation: State of the art

- The **CoNLL-X (2006) shared task** provides evaluation numbers for various dependency parsing approaches over 13 languages

Parser	UAS%
Sagae and Lavie (2006) ensemble of dependency parsers	92.7
Charniak (2000) generative, constituency	92.2
Collins (1999) generative, constituency	91.7
McDonald and Pereira (2005) – MST graph-based dependency	91.5
Yamada and Matsumoto (2003) – transition-based dependency	90.4

Evaluation: State of the art

- Neural Approaches: **Neural Span Parser**



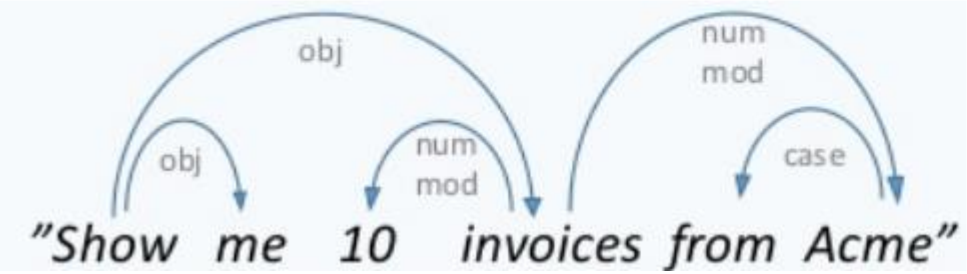
Evaluation: State of the art

- Neural Approaches: **Neural Span Parser**

Parser	F
Automatic splitting (Petrov 2006)	89.6
Non-neural re-ranking parser (Charniak 2005)	91.0
Neural span parser (Stern 2017)	91.7
State-of-the-art neural (Kitaev 2018)	93.6
+ extra unlabeled data (Kitaev 2018)	95.1

Why Parsing

- **Chatbot**
  - Identify root of the user request
  - Extract related objects

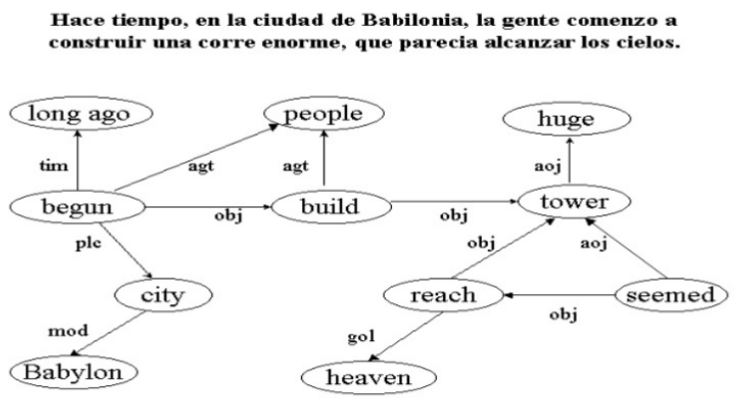


show(invoices) → invoices = ACME

→ *num*(invoices) = 10

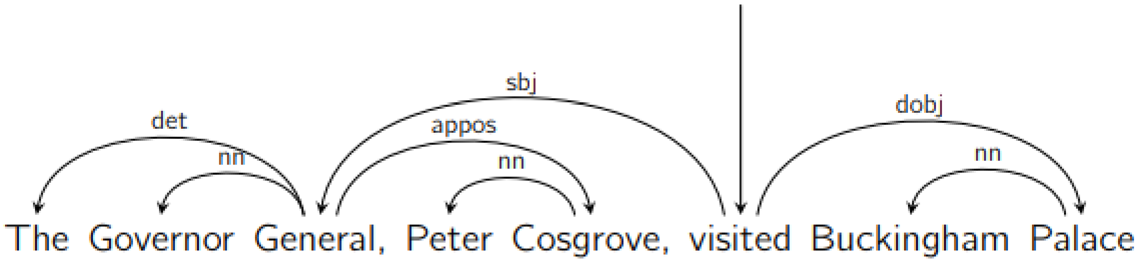
Why Parsing

- Chatbot
- **Machine Translation**
  - Extract a language-agnostic representation such as UNL http://www.unlweb.net/wiki/index.php/Introduction_to_UNL



Why Parsing

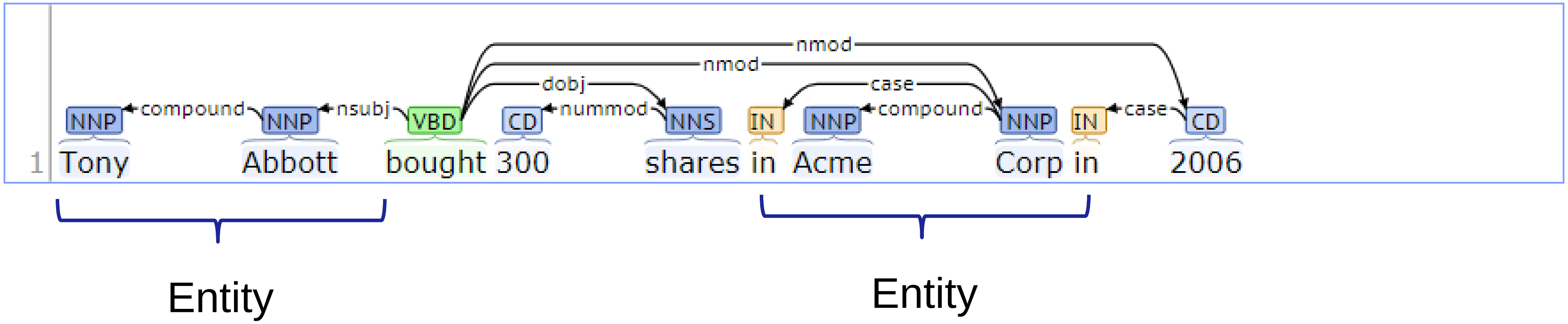
- **Chatbot**
- **Machine Translation**
- **Relation Extraction**



Who is the governor general?

Why Parsing

- Chatbot
- Machine Translation
- Relation Extraction
- **Enhance NER**



Why Parsing

- **Chatbot**
- **Machine Translation**
- **Relation Extraction**
- **Enhance NER**
- **Many more...**
  - A survey of parsing and its applications: http://web.science.mq.edu.au/~mjohnson/papers/Johnson15ParsingSurvey.pdf

Practicals

- NLTK Parsers are outdated and slow
- Use Stanford Parser
  - Download CoreNLP server: https://stanfordnlp.github.io/CoreNLP/corenlp-server.html

```
from nltk.parse.corenlp import CoreNLPDependencyParser
dep_parser = CoreNLPDependencyParser(url='http://localhost:9000')
```

```
parse, = dep_parser.raw_parse('The quick brown fox jumps over the lazy dog.')
print(parse.tree())
```

```
The     DT    4   det
quick   JJ    4   amod
brown   JJ    4   amod
fox     NN    5   nsubj
jumps   VBZ   0   ROOT
over    IN    9   case
the     DT    9   det
lazy    JJ    9   amod
dog     NN    5   nmod
.       .     5   punct
```

Practicals

- NLTK Parser are outdate and slow
- Use Stanford Parser
- or spacy
  - https://spacy.io/usage/linguistic-features#dependency-parse

```
import spacy

nlp = spacy.load('en')
doc = nlp('The quick brown fox jumps over the lazy dog.')
```

  - Practical examples: https://shirishkadam.com/2016/12/23/dependency-parsing-in-nlp/