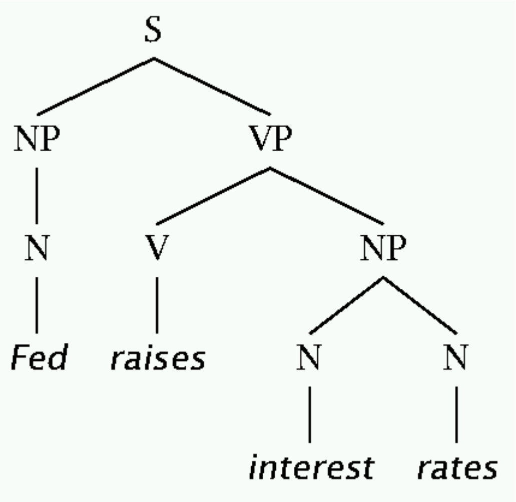
Statistical Natural Language Parsing

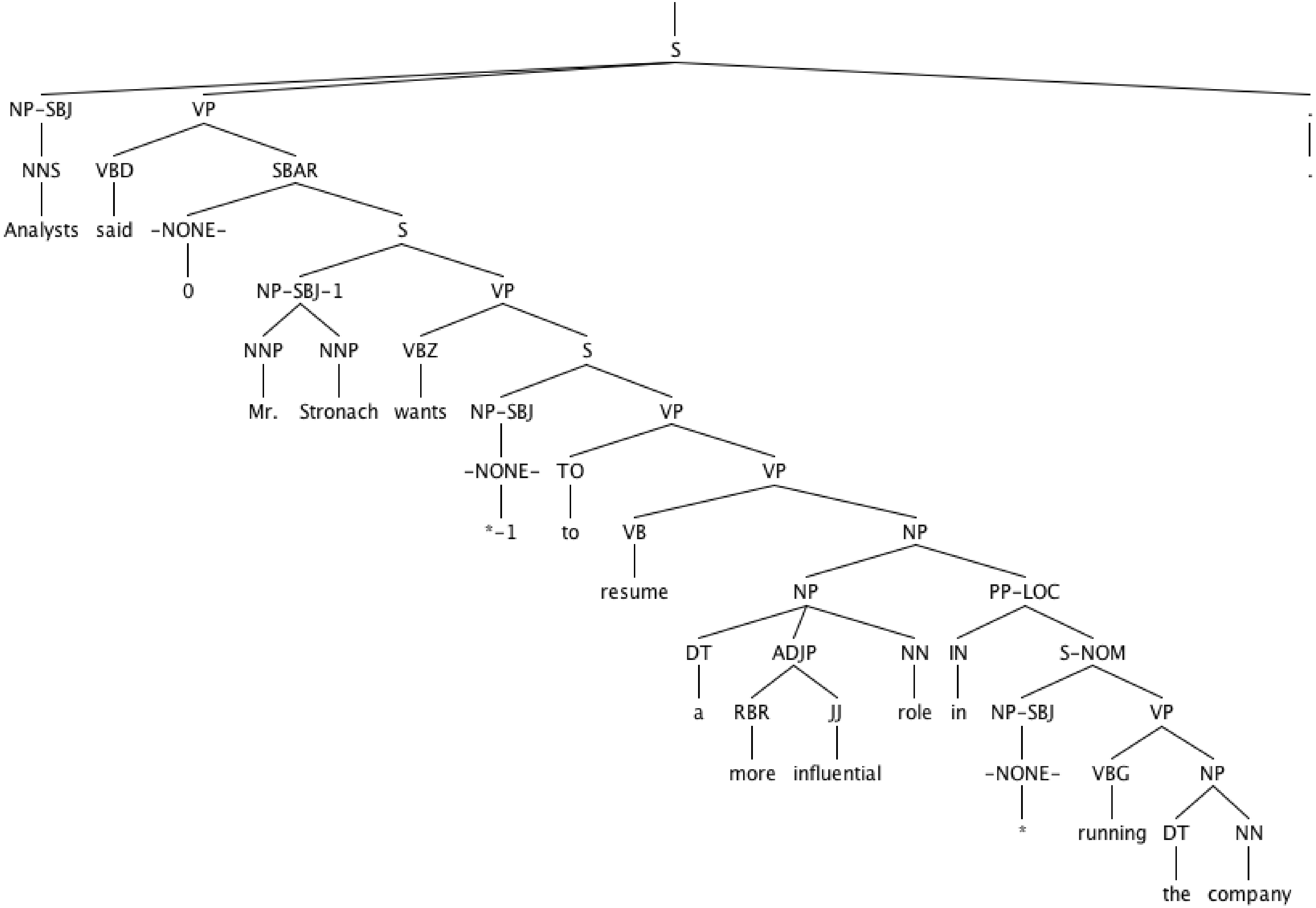
Mausam

(Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Luke Zettlemoyer)

Two views of linguistic structure:
1. Constituency (phrase structure)

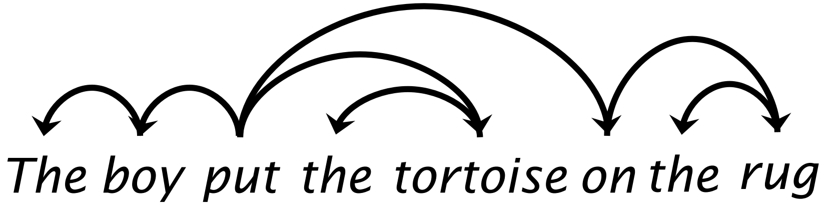
- Phrase structure organizes words into nested constituents.
- How do we know what is a constituent? (Not that linguists don't argue about some cases.)
  - Distribution: a constituent behaves as a unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about
  - Substitution/expansion/pro-forms:
    - I sat [on the box/right on top of the box/there].
  - Coordination, regular internal structure, no intrusion, fragments, semantics, …

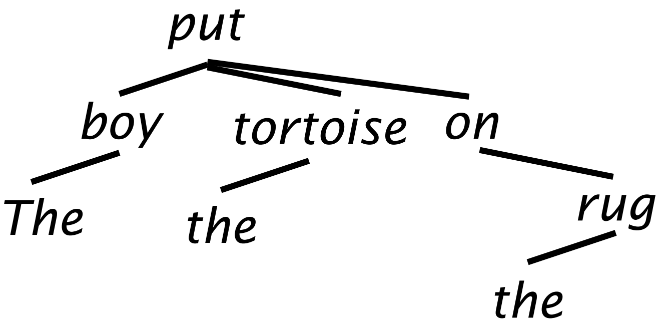



S
NP-SBJ
VP
.
NNS
VBD
SBAR
.
Analysts
said
-NONE-
S
0
NP-SBJ-1
VP
NNP
NNP
VBZ
S
Mr.
Stronach
wants
NP-SBJ
VP
-NONE-
TO
VP
*-1
to
VB
NP
resume
NP
PP-LOC
DT
ADJP
NN
IN
S-NOM
a
RBR
JJ
role
in
NP-SBJ
VP
more
influential
-NONE-
VBG
NP
*
running
DT
NN
the
company

Two views of linguistic structure: 2. Dependency structure

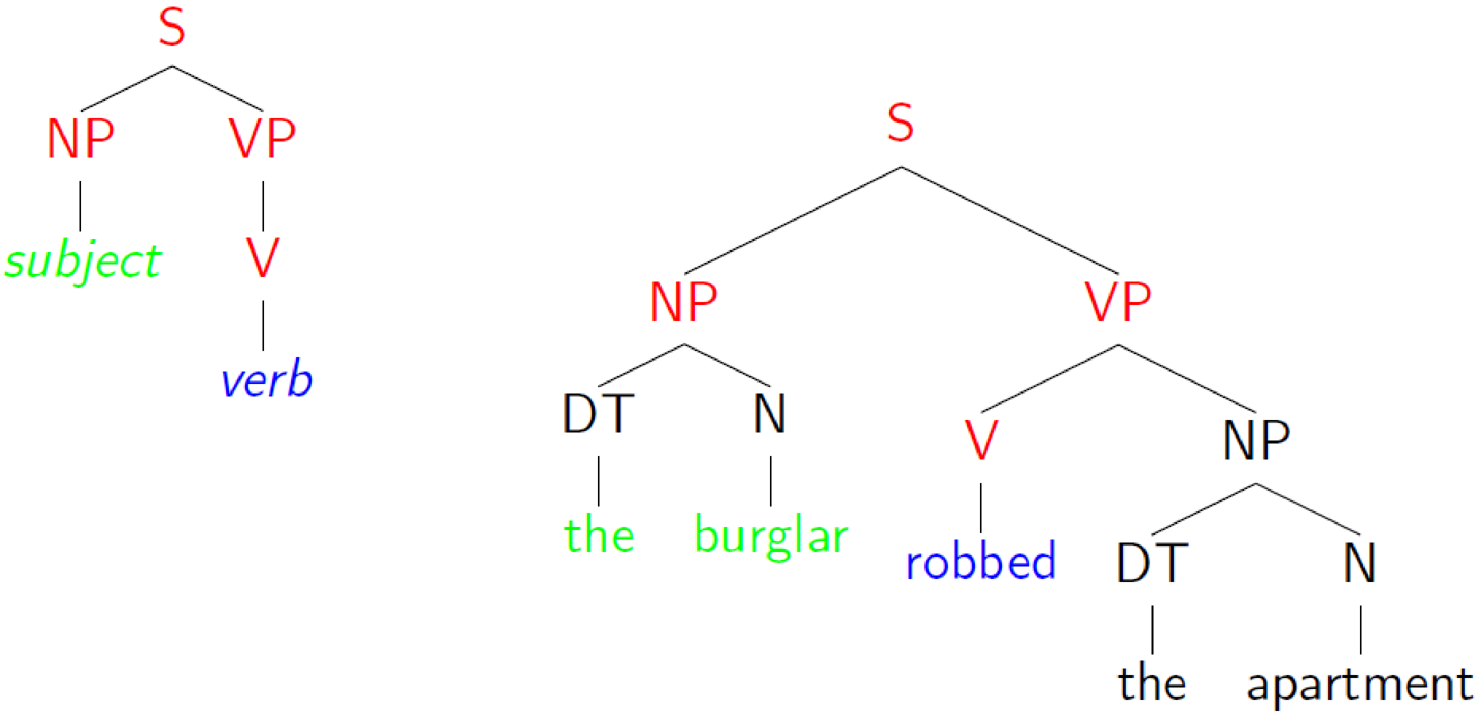
- Dependency structure shows which words depend on (modify or are arguments of) which other words.





Why Parse?

- Part of speech information
- Phrase information
- Useful relationships



⇒ "the burglar" is the subject of "robbed"

The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, *Computational Linguistics*]

```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```

Penn Treebank Non-terminals

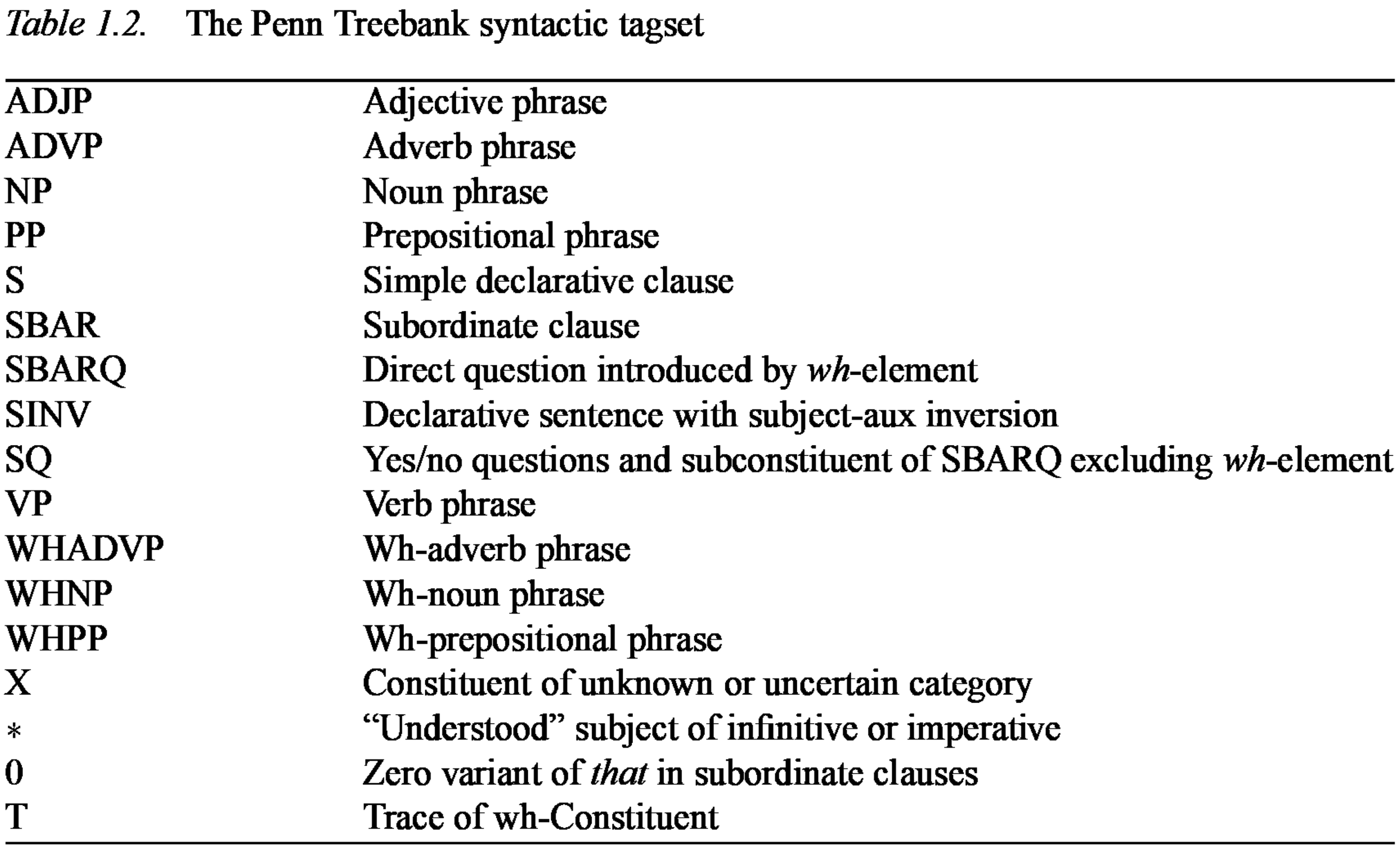


Table 1.2. The Penn Treebank syntactic tagset

ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Subordinate clause
SBARQ	Direct question introduced by *wh*-element
SINV	Declarative sentence with subject-aux inversion
SQ	Yes/no questions and subconstituent of SBARQ excluding *wh*-element
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category
*	"Understood" subject of infinitive or imperative
0	Zero variant of *that* in subordinate clauses
T	Trace of wh-Constituent

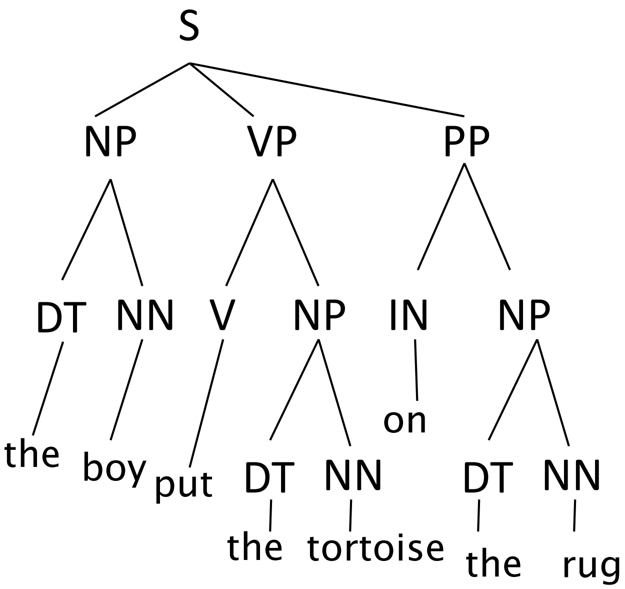
Statistical parsing applications

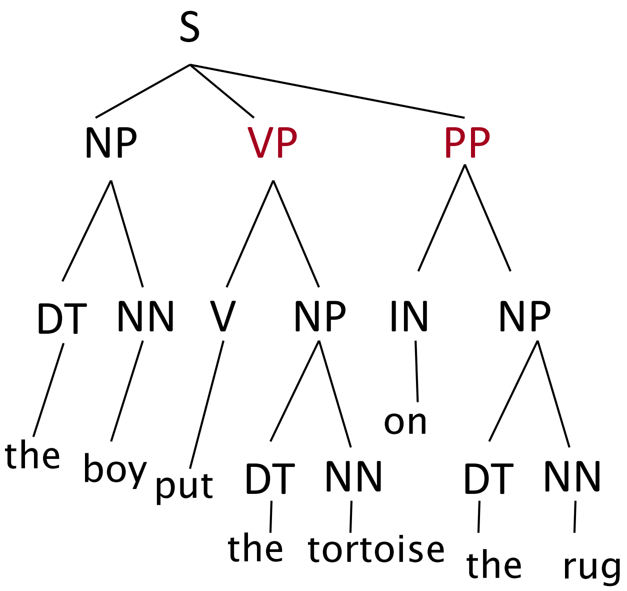
Statistical parsers are now robust and widely used in larger NLP applications:

- High precision question answering [Pasca and Harabagiu SIGIR 2001]
- Improving biological named entity finding [Finkel et al. JNLPBA 2004]
- Syntactically based sentence compression [Lin and Wilbur 2007]
- Extracting opinions about products [Bloom et al. NAACL 2007]
- Improved interaction in computer games [Gorniak and Roy 2005]
- Helping linguists find data [Resnik et al. BLS 2005]
- Source sentence analysis for machine translation [Xu et al. 2009]
- Relation extraction systems [Fundel et al. *Bioinformatics* 2006]

Example Application: Machine Translation

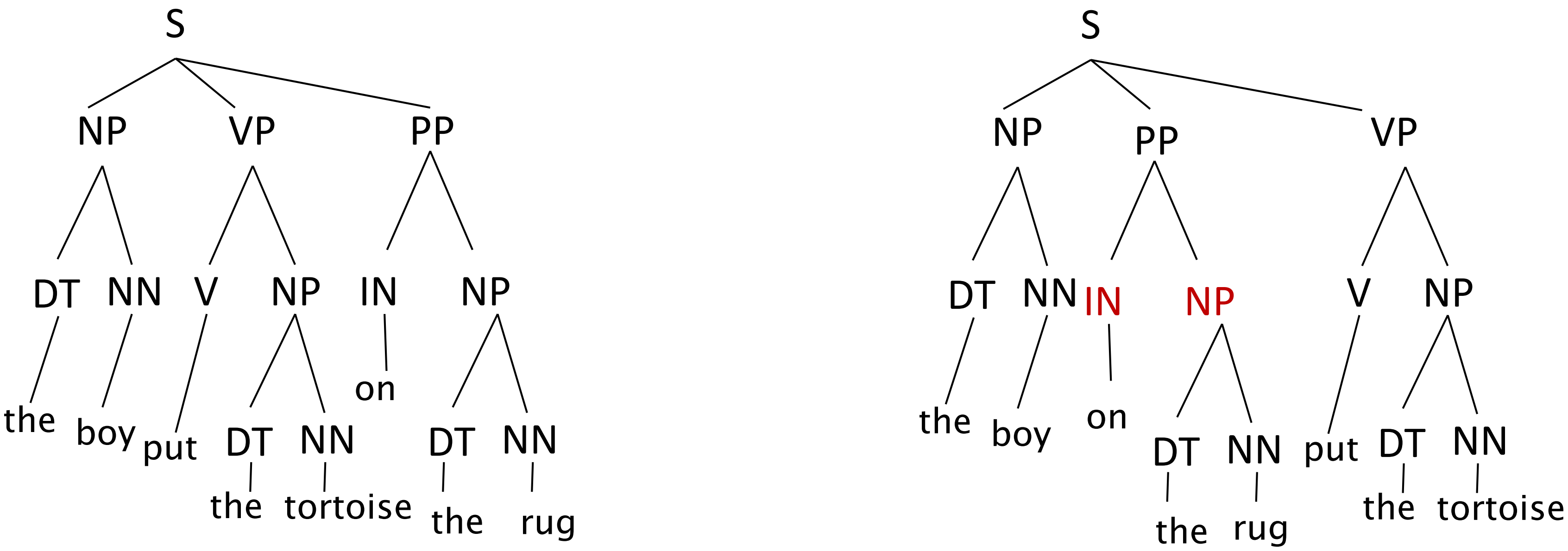
- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position





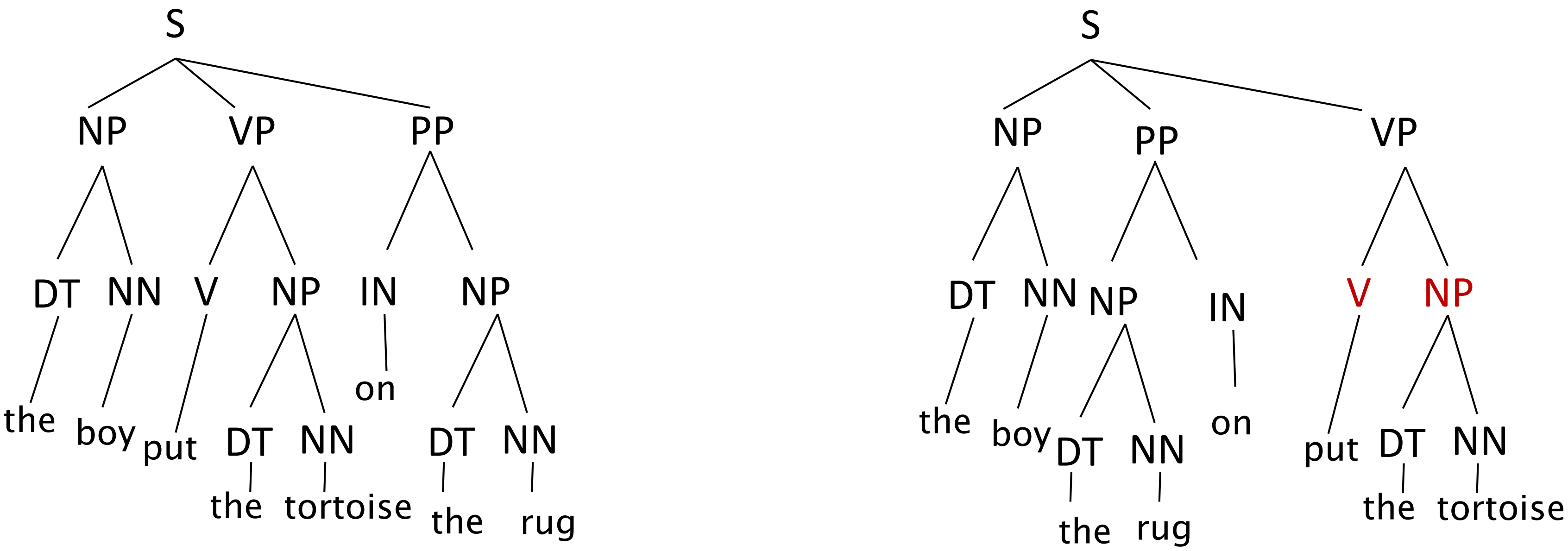
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position



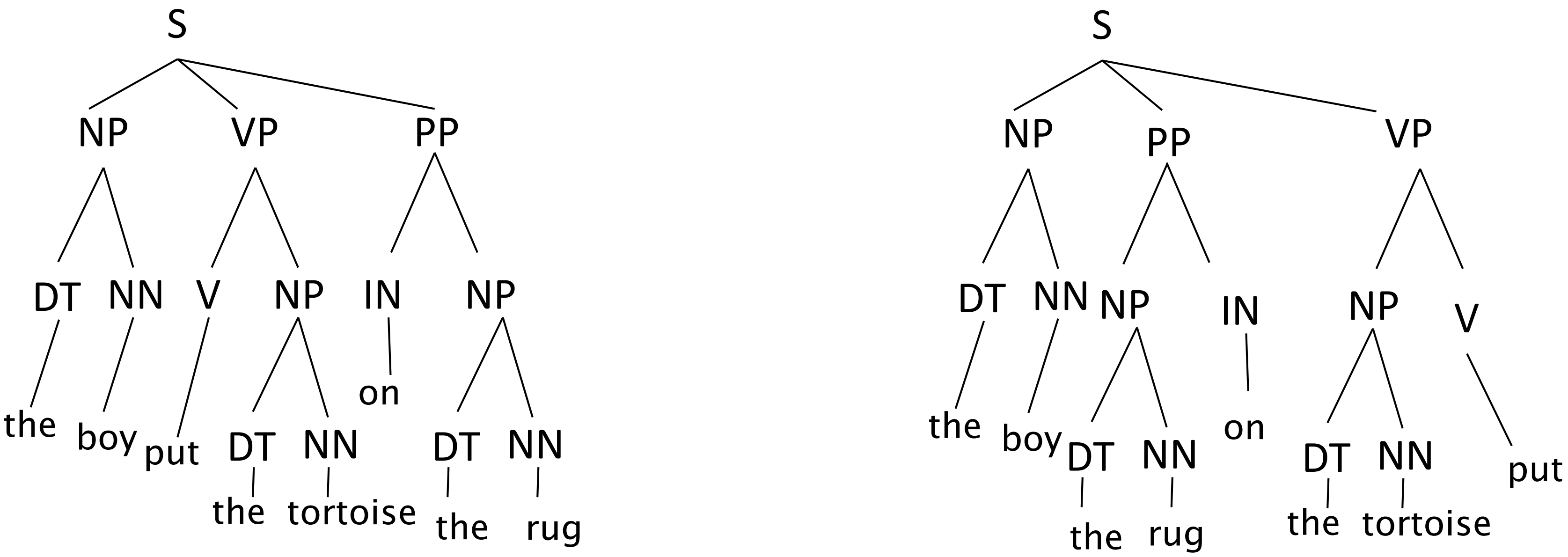
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position



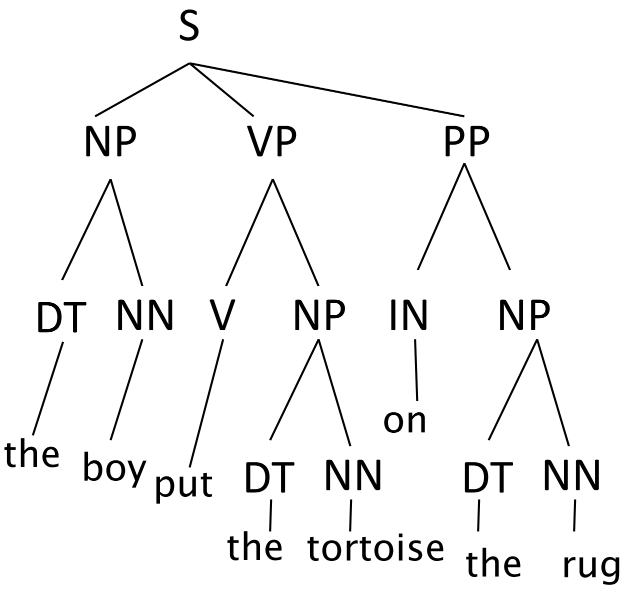
Example Application: Machine Translation

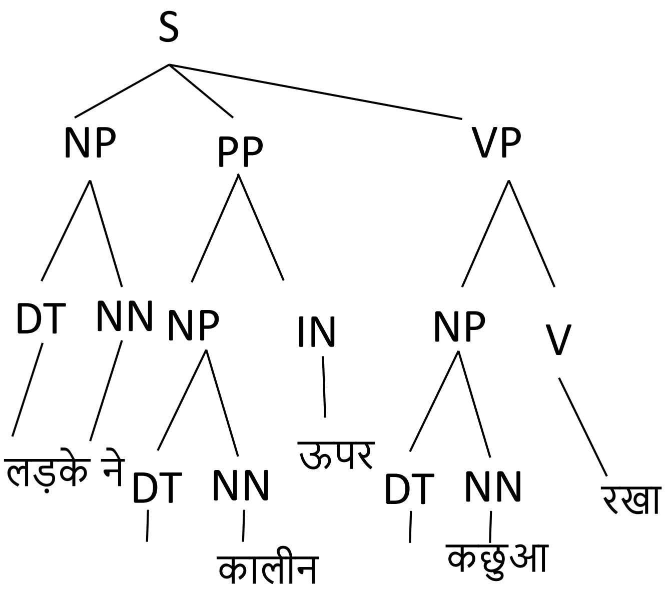
- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position



Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position





Pre 1990 (“Classical”) NLP Parsing

- Goes back to Chomsky’s PhD thesis in 1950s
- Wrote symbolic grammar (CFG or often richer) and lexicon

  S → NP VP
  NP → (DT) NN
  NP → NN NNS
  NP → NNP
  VP → V NP

  NN → *interest*
  NNS → *rates*
  NNS → *raises*
  VBP → *interest*
  VBZ → *rates*

- Used grammar/proof systems to prove parses from words
- This scaled very badly and didn’t give coverage. For sentence:

  Fed raises interest rates 0.5% in effort to control inflation

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

Classical NLP Parsing:
The problem and its solution

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
  - But the attempt make the grammars not robust
    - In traditional systems, commonly 30% of sentences in even an edited text would have *no* parse.
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find ***the most likely parse(s)*** for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)

Context Free Grammars and Ambiguities

Context-Free Grammars

Hopcroft and Ullman, 1979

A context free grammar $G = (N, \Sigma, R, S)$ where:

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules of the form $X \rightarrow Y_1 Y_2 \ldots Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
- $S \in N$ is a distinguished start symbol

Context-Free Grammars in NLP

- A context free grammar G in NLP = (N, C, Σ, S, L, R)
  - Σ is a set of terminal symbols
  - C is a set of preterminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - L is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in C$ and $x \in \Sigma$
  - R is the grammar, a set of items of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup C)^*$
- By usual convention, S is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write *e* for an empty sequence, rather than nothing

A Context Free Grammar of English

$$N = \{\text{S, NP, VP, PP, DT, Vi, Vt, NN, IN}\}$$
$$S = \text{S}$$
$$\Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\}$$

$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
DT	→	the
IN	→	with
IN	→	in

Note: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

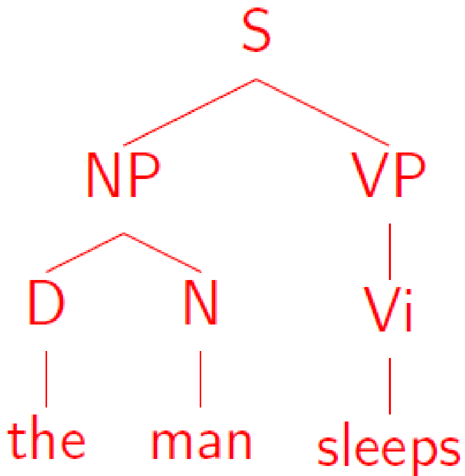
Left-Most Derivations

A left-most derivation is a sequence of strings $s_1 \ldots s_n$, where

- $s_1 = S$, the start symbol
- $s_n \in \Sigma^*$, i.e. s_n is made up of terminal symbols only
- Each s_i for $i = 2 \ldots n$ is derived from s_{i-1} by picking the left-most non-terminal X in s_{i-1} and replacing it by some β where $X \rightarrow \beta$ is a rule in R

For example: [S], [NP VP], [D N VP], [the N VP], [the man VP], [the man Vi], [the man sleeps]

Representation of a derivation as a tree:



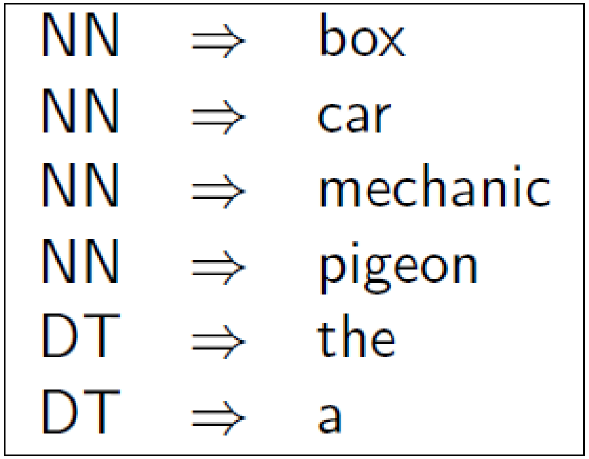
Properties of CFGs

- A CFG defines a set of possible derivations

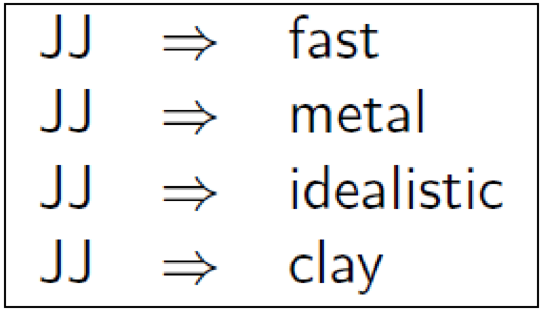
- A string $s \in \Sigma^*$ is in the *language* defined by the CFG if there is at least one derivation that yields s

- Each string in the language generated by the CFG may have more than one derivation (“ambiguity”)

A Fragment of a Noun Phrase Grammar



NN	⇒	box
NN	⇒	car
NN	⇒	mechanic
NN	⇒	pigeon
DT	⇒	the
DT	⇒	a



JJ	⇒	fast
JJ	⇒	metal
JJ	⇒	idealistic
JJ	⇒	clay

Extended Grammar with Prepositional Phrases

$\bar{N}$	⇒	NN	
$\bar{N}$	⇒	NN	$\bar{N}$
$\bar{N}$	⇒	JJ	$\bar{N}$
$\bar{N}$	⇒	$\bar{N}$	$\bar{N}$
NP	⇒	DT	$\bar{N}$

NN	⇒	box
NN	⇒	car
NN	⇒	mechanic
NN	⇒	pigeon
DT	⇒	the
DT	⇒	a

JJ	⇒	fast
JJ	⇒	metal
JJ	⇒	idealistic
JJ	⇒	clay
IN	⇒	in
IN	⇒	under
IN	⇒	of
IN	⇒	on
IN	⇒	with
IN	⇒	as

Generates:

in a box, under the box, the fast car mechanic under the pigeon in the box, . . .

Verbs, Verb Phrases and Sentences

- Basic Verb Types

  Vi = Intransitive verb e.g., sleeps, walks, laughs

  Vt = Transitive verb e.g., sees, saw, likes

  Vd = Ditransitive verb e.g., gave

- Basic VP Rules

  VP → Vi

  VP → Vt NP

  VP → Vd NP NP

- Basic S Rule

  S → NP VP

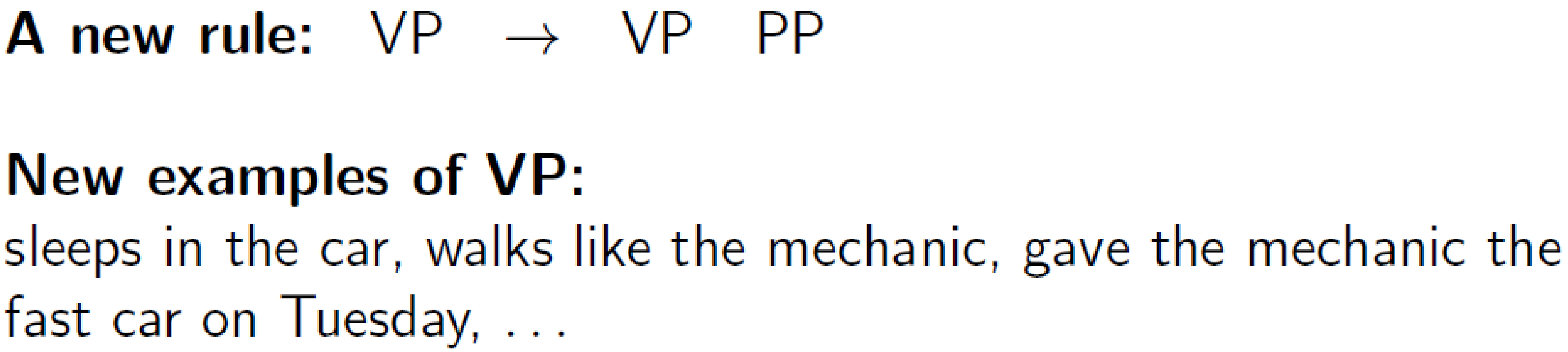
Examples of VP:

sleeps, walks, likes the mechanic, gave the mechanic the fast car

Examples of S:

the man sleeps, the dog walks, the dog gave the mechanic the fast car

PPs Modifying Verb Phrases



A new rule: VP → VP PP

New examples of VP:
sleeps in the car, walks like the mechanic, gave the mechanic the fast car on Tuesday, . . .

Complementizers and SBARs

- Complementizers

  COMP = complementizer e.g., that

- SBAR

  SBAR → COMP S

Examples:

that the man sleeps, that the mechanic saw the dog . .

More Verbs

- New Verb Types

V[5]	e.g., said, reported
V[6]	e.g., told, informed
V[7]	e.g., bet

- New VP Rules

VP	→	V[5]	SBAR		
VP	→	V[6]	NP	SBAR	
VP	→	V[7]	NP	NP	SBAR

Examples of New VPs:

said that the man sleeps

told the dog that the mechanic likes the pigeon

bet the pigeon $50 that the mechanic owns a fast car

Coordination

- A New Part-of-Speech:
  CC = Coordinator   e.g., and, or, but

- New Rules

$$\begin{array}{lllll}
\text{NP} & \rightarrow & \text{NP} & \text{CC} & \text{NP} \\
\bar{\text{N}} & \rightarrow & \bar{\text{N}} & \text{CC} & \bar{\text{N}} \\
\text{VP} & \rightarrow & \text{VP} & \text{CC} & \text{VP} \\
\text{S} & \rightarrow & \text{S} & \text{CC} & \text{S} \\
\text{SBAR} & \rightarrow & \text{SBAR} & \text{CC} & \text{SBAR}
\end{array}$$

Much more remains…

- Agreement

  The dogs laugh *vs.* The dog laughs

- Wh-movement

  The dog that the cat liked ___

- Active vs. passive

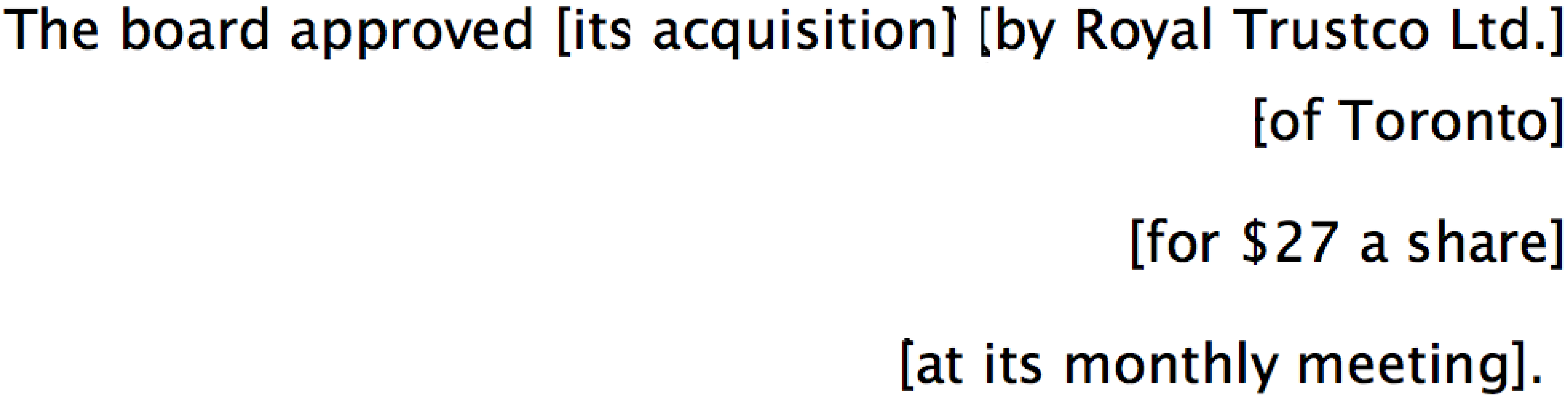
  The dog saw the cat *vs.*
  The cat was seen by the dog

- If you're interested in reading more:

  Syntactic Theory: A Formal Introduction, 2nd Edition. Ivan A. Sag, Thomas Wasow, and Emily M. Bender.

Attachment ambiguities

- A key parsing decision is how we 'attach' various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.



The board approved [its acquisition] [by Royal Trustco Ltd.]

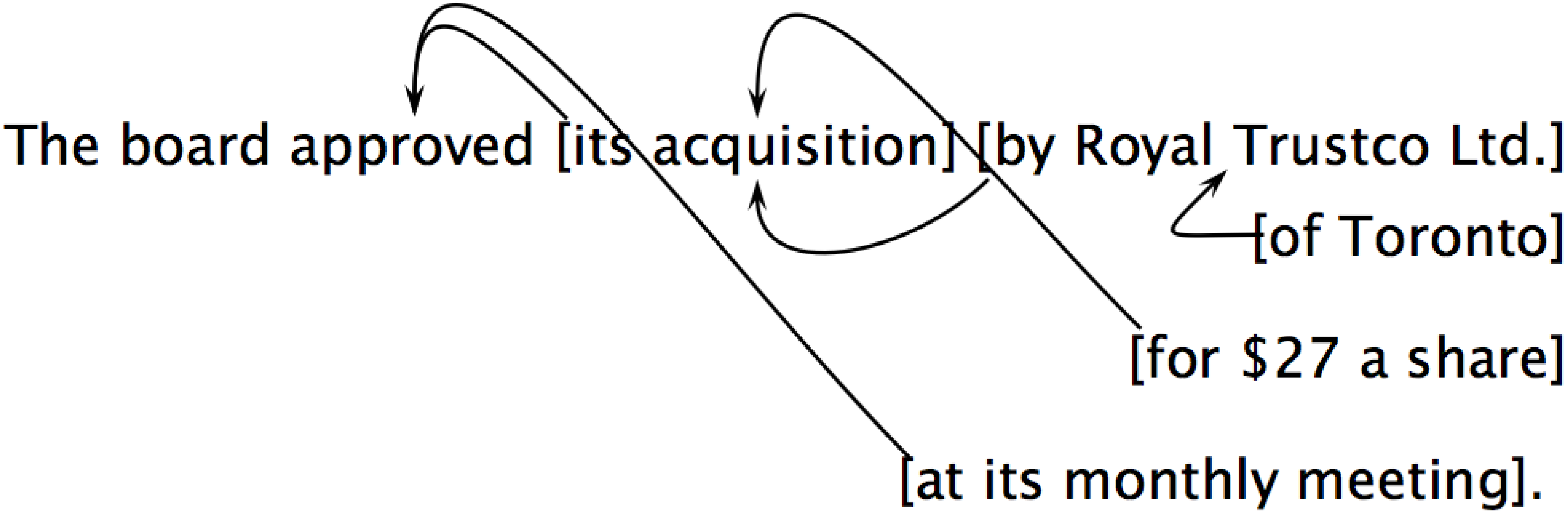
[of Toronto]

[for $27 a share]

[at its monthly meeting].

Attachment ambiguities

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.



The board approved [its acquisition] [by Royal Trustco Ltd.]

[of Toronto]

[for $27 a share]

[at its monthly meeting].

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with n+2 sides
    - Turns up in triangulation of probabilistic graphical models….

Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink

Syntactic Ambiguities I

- Prepositional phrases:
  They cooked the beans in the pot on the stove with handles.
- Particle vs. preposition:
  The lady dressed up the staircase.
- Complement structures
  The tourists objected to the guide that they couldn't hear.
  She knows you like the back of her hand.
- Gerund vs. participial adjective
  Visiting relatives can be boring.
  Changing schedules frequently confused passengers.

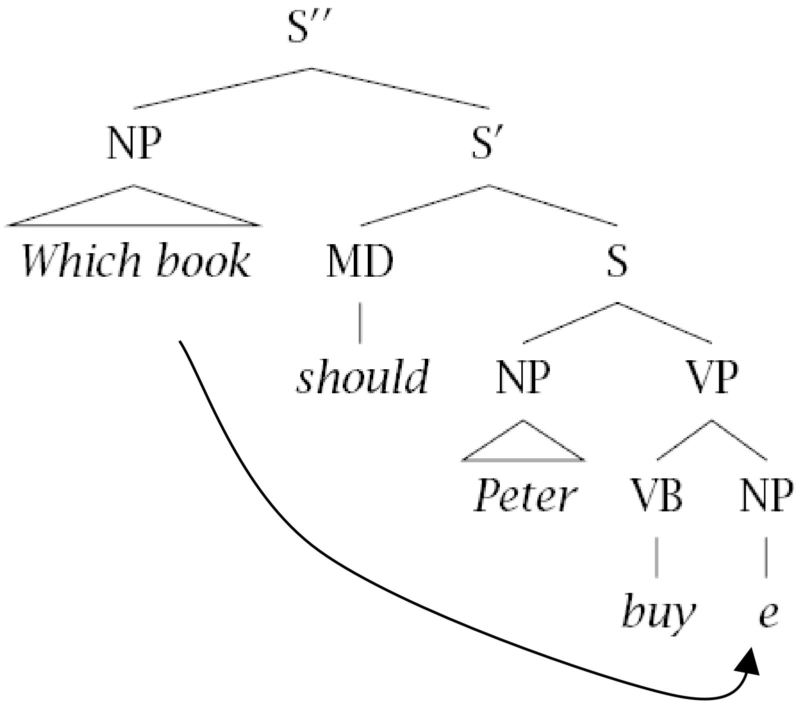
Syntactic Ambiguities II

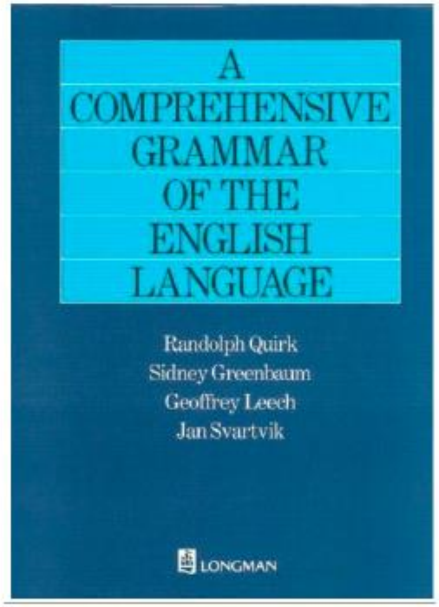
- Modifier scope within NPs
  impractical design requirements
  plastic cup holder
- Multiple gap constructions
  The chicken is ready to eat.
  The contractors are rich enough to sue.
- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.

Non-Local Phenomena

- Dislocation / gapping
  - Which book should Peter buy?
  - A debate arose which continued until the election.

- Binding
  - Reference
    - The IRS audits itself
  - Control
    - I want to go
    - I want you to go



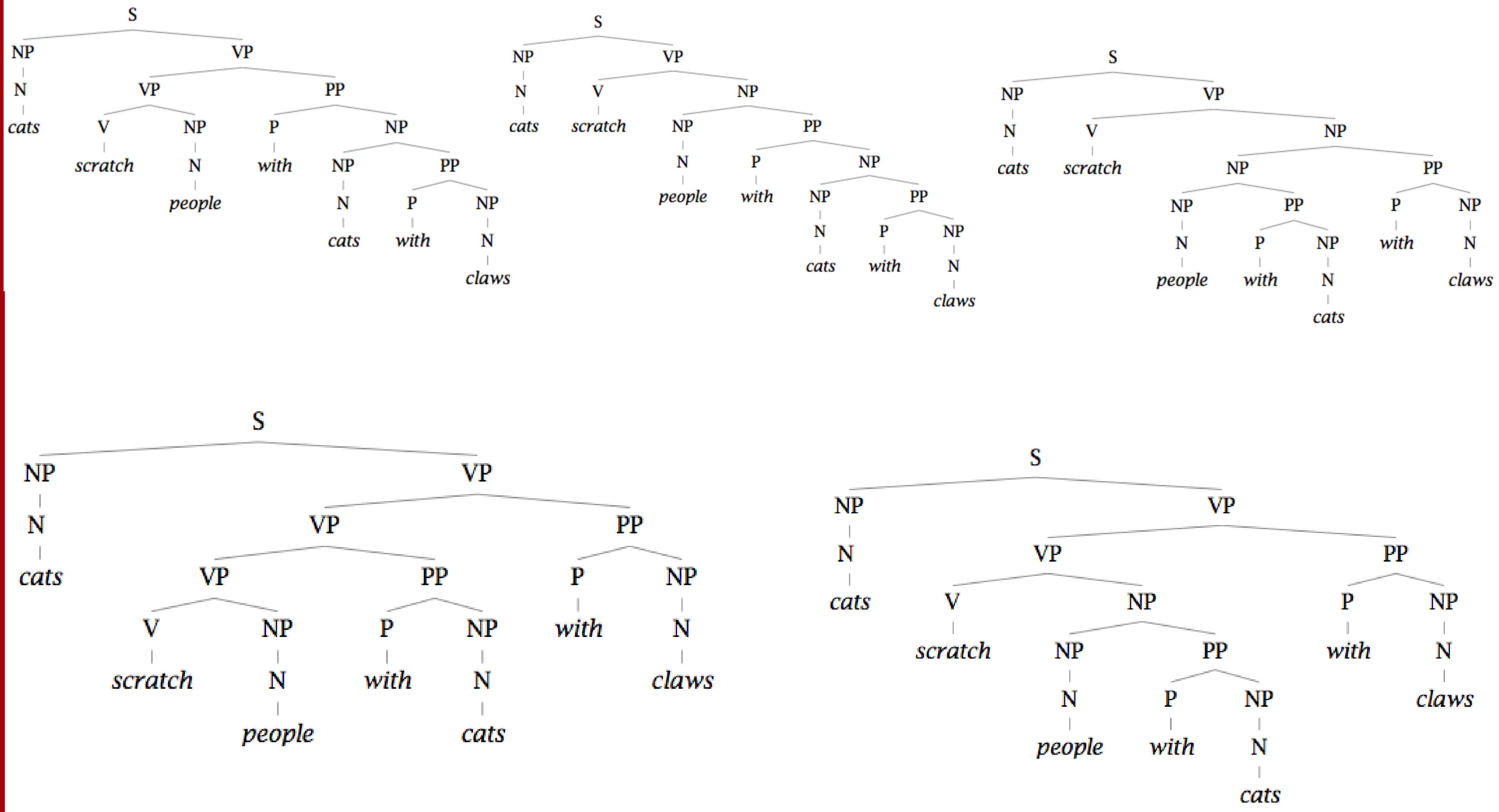


Product Details (from Amazon)
Hardcover: 1779 pages
Publisher: Longman; 2nd Revised edition
Language: English
ISBN-10: 0582517346
ISBN-13: 978-0582517349
Product Dimensions: 8.4 x 2.4 x 10 inches
Shipping Weight: 4.6 pounds

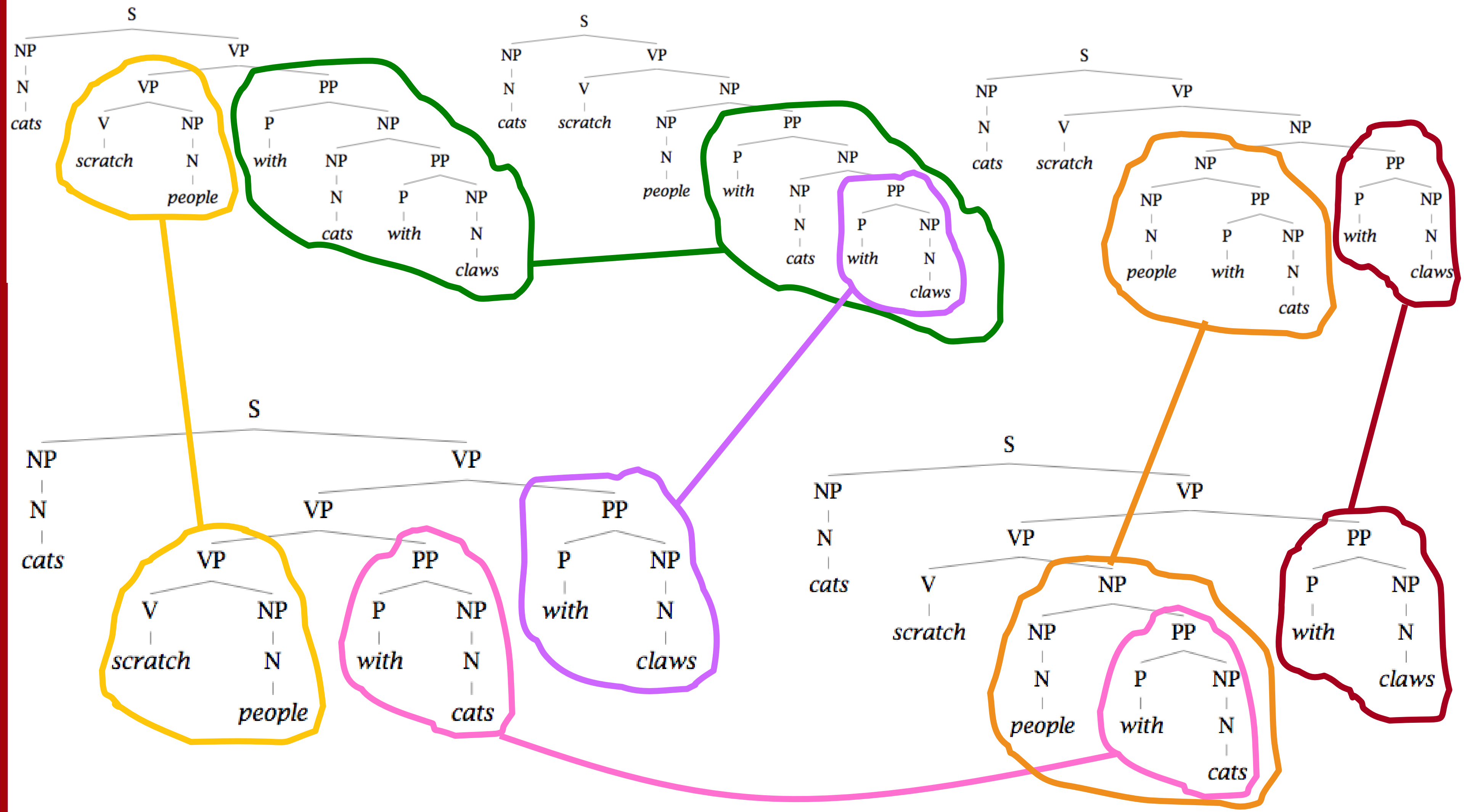
Context-Free Grammars in NLP

- A context free grammar G in NLP = (N, C, Σ, S, L, R)
  - Σ is a set of terminal symbols
  - C is a set of preterminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - L is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in C$ and $x \in \Sigma$
  - R is the grammar, a set of items of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup C)^*$
- By usual convention, S is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write *e* for an empty sequence, rather than nothing

Parsing: Two problems to solve:
1. Repeated work…



Parsing: Two problems to solve:
1. Repeated work…



Parsing: Two problems to solve:
2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
- Is the problem ‘AI complete’? Yes, but …
- Words are good predictors of attachment
    - Even absent full understanding
  - Moscow sent more than 100,000 soldiers into Afghanistan …
  - Sydney Water breached an agreement with NSW Health …
- Our statistical parsers will try to exploit such statistics.

Probabilistic Context Free Grammar

Probabilistic – or stochastic – context-free grammars (PCFGs)

- G = (Σ, N, S, R, P)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - R is a set of rules/productions of the form X → γ
  - P is a probability function
    - P: R → [0,1]
    - $\forall X \in N, \sum_{X \to \gamma \in R} P(X \to \gamma) = 1$
- A grammar G generates a language model L.

$$\text{å}_{g\text{Î}\ T*} P(g) = 1$$

PCFG Example

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

- Probability of a tree t with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$$

is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

Example of a PCFG

S	$\Rightarrow$	NP	VP	1.0
VP	$\Rightarrow$	Vi		0.4
VP	$\Rightarrow$	Vt	NP	0.4
VP	$\Rightarrow$	VP	PP	0.2
NP	$\Rightarrow$	DT	NN	0.3
NP	$\Rightarrow$	NP	PP	0.7
PP	$\Rightarrow$	P	NP	1.0

Vi	$\Rightarrow$	sleeps	1.0
Vt	$\Rightarrow$	saw	1.0
NN	$\Rightarrow$	man	0.7
NN	$\Rightarrow$	woman	0.2
NN	$\Rightarrow$	telescope	0.1
DT	$\Rightarrow$	the	1.0
IN	$\Rightarrow$	with	0.5
IN	$\Rightarrow$	in	0.5

- Probability of a tree t with rules

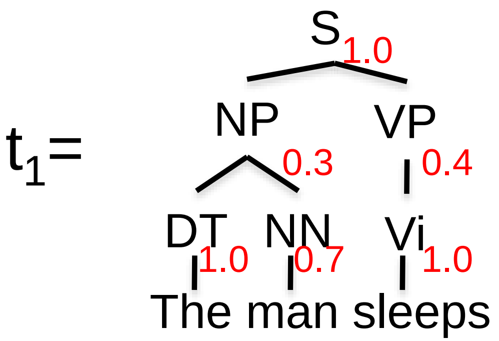
$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$$

is $p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$ where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

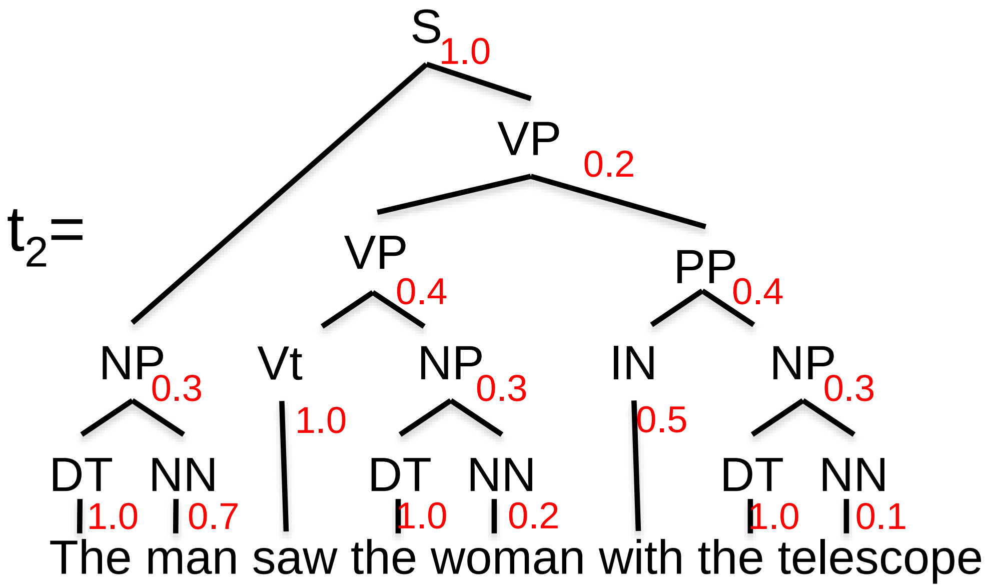
Probability of a Parse

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5



$p(t_1)$=1.0*0.3*1.0*0.7*0.4*1.0



$p(t_s)$=1.8*0.3*1.0*0.7*0.2*0.4*1.0*0.3*1.0*0.2*0.4*0.5*0.3*1.0*0.1

PCFGs: Learning and Inference

- Model
  - The probability of a tree t with n rules α_i → β_i, i = 1..n

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

- Learning
  - Read the rules off of labeled sentences, use ML estimates for probabilities

$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

  - and use all of our standard smoothing tricks!

- Inference
  - For input sentence s, define T(s) to be the set of trees whose *yield* is s (whole leaves, read left to right, match the words in s)

$$t^*(s) = \arg\max_{t \in \mathcal{T}(s)} p(t)$$

Grammar Transforms

Chomsky Normal Form

- All rules are of the form $X \to Y\ Z$ or $X \to w$
  - $X, Y, Z \in N$ and $w \in \Sigma$
- A transformation to this form doesn’t change the weak generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals ($n > 2$)

A phrase structure grammar

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Chomsky Normal Form steps

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 steps

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

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

VP → 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

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

S → *people*

V → *fish*

S → *fish*

V → *tanks*

S → *tanks*

P → *with*

Chomsky Normal Form steps

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

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

S → *people*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

Chomsky Normal Form steps

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 PP

NP → P NP

PP → P NP

NP → *people*

NP → *fish*

NP → *tanks*

NP → *rods*

V → *people*

S → *people*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

PP → *with*

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

VP → V @VP_V

@VP_V → NP PP

S → V @S_V

@S_V → NP PP

VP → V PP

S → V PP

NP → NP NP

NP → NP PP

NP → P NP

PP → P NP

NP → *people*

NP → *fish*

NP → *tanks*

NP → *rods*

V → *people*

S → *people*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

PP → *with*

A phrase structure grammar

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Chomsky Normal Form steps

S → NP VP

VP → V NP

S → V NP

VP → V @VP_V

@VP_V → NP PP

S → V @S_V

@S_V → NP PP

VP → V PP

S → V PP

NP → NP NP

NP → NP PP

NP → P NP

PP → P NP

NP → *people*

NP → *fish*

NP → *tanks*

NP → *rods*

V → *people*

S → *people*

VP → *people*

V → *fish*

S → *fish*

VP → *fish*

V → *tanks*

S → *tanks*

VP → *tanks*

P → *with*

PP → *with*

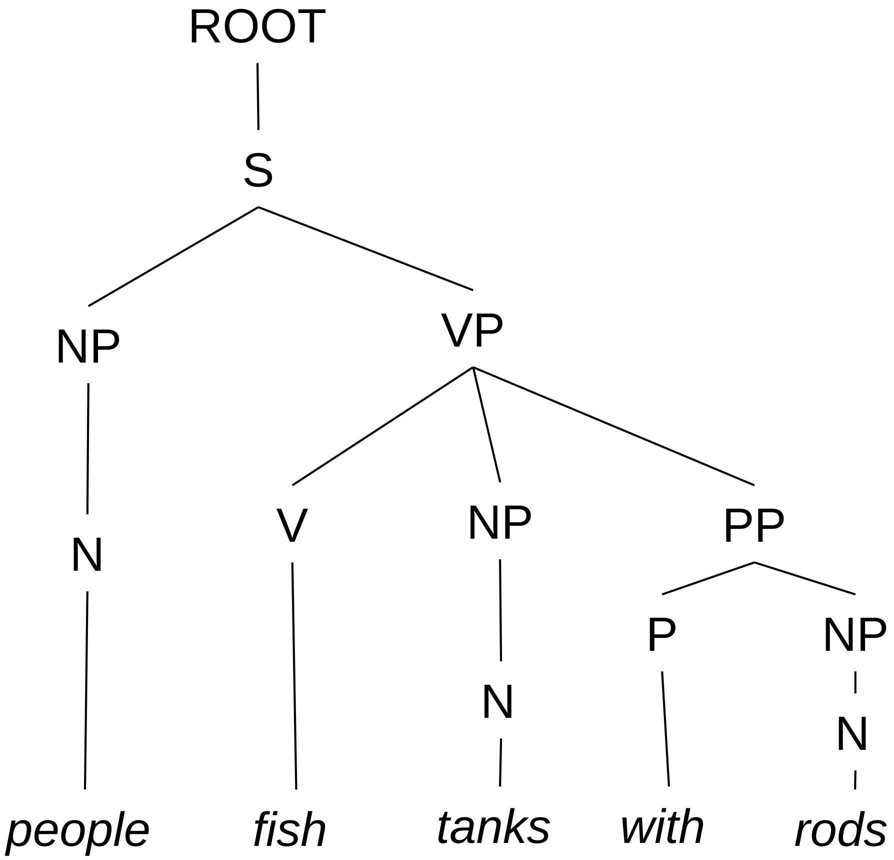
Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

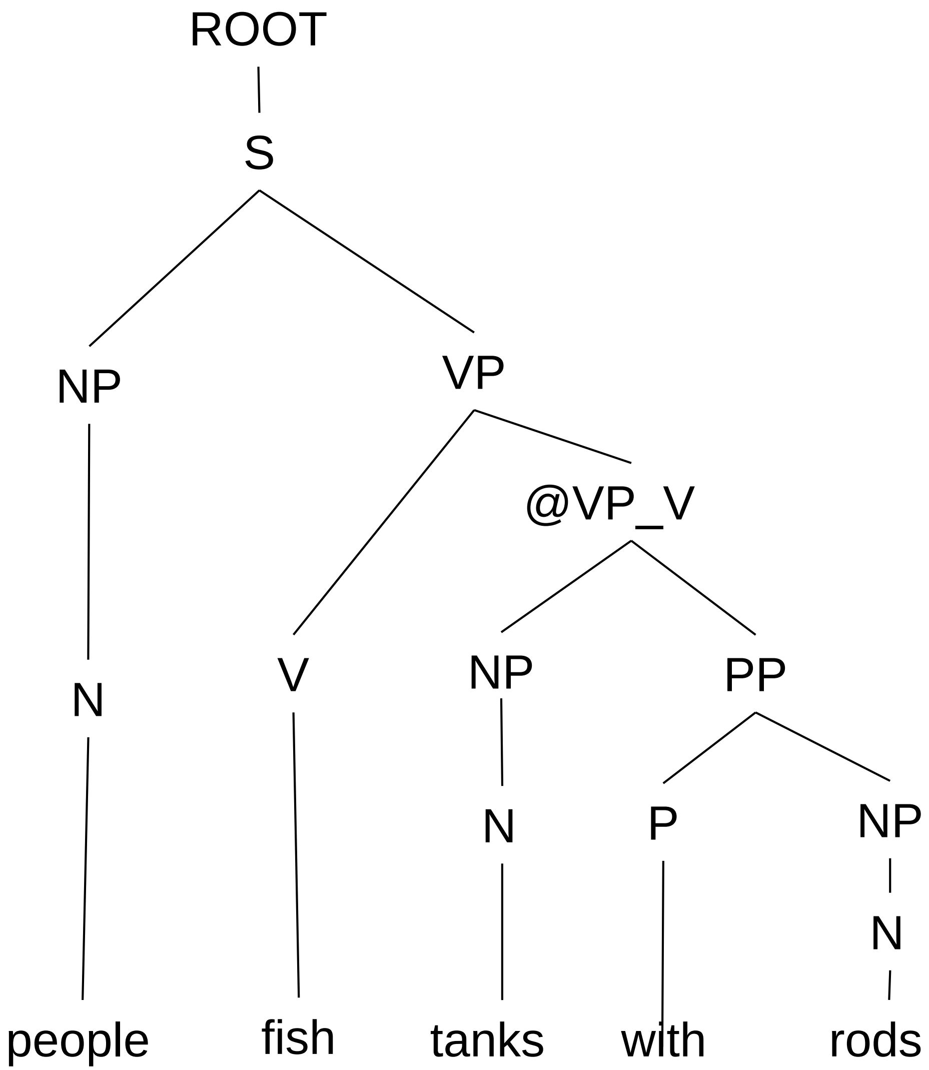
- **Binarization** is crucial for cubic time CFG parsing

- The rest isn’t necessary; it just makes the algorithms cleaner and a bit quicker

An example: before binarization…

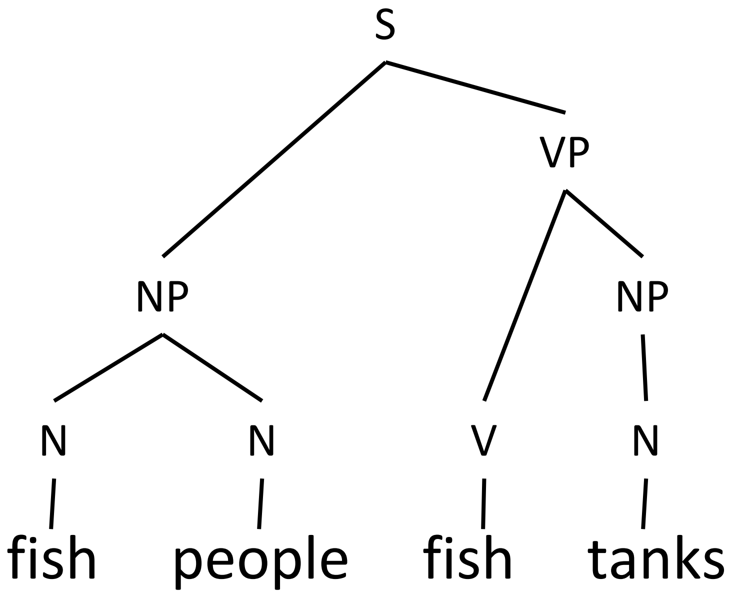


After binarization…



Parsing

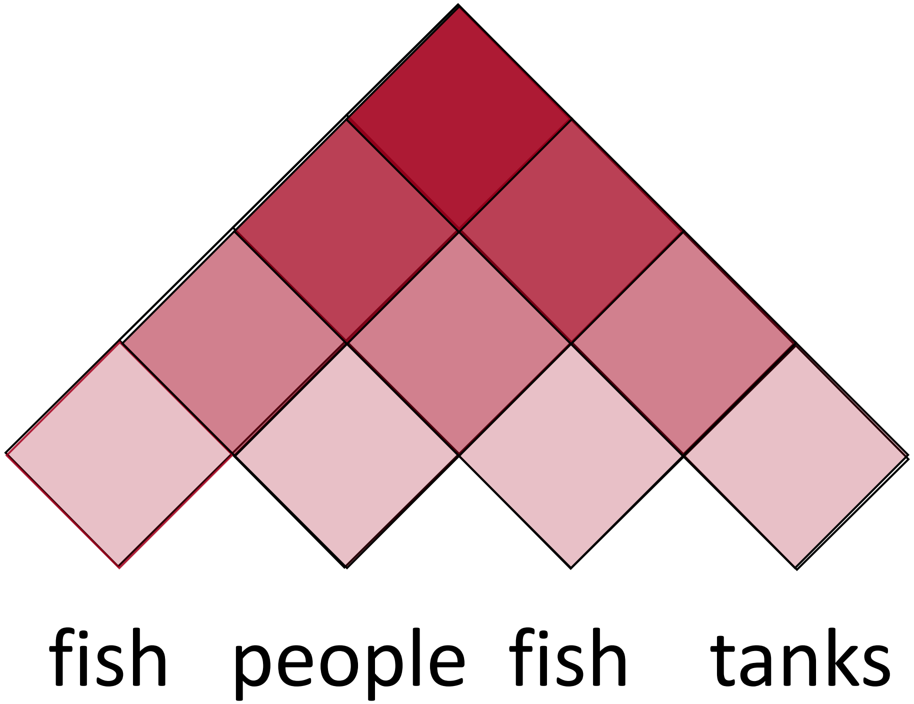
Constituency Parsing



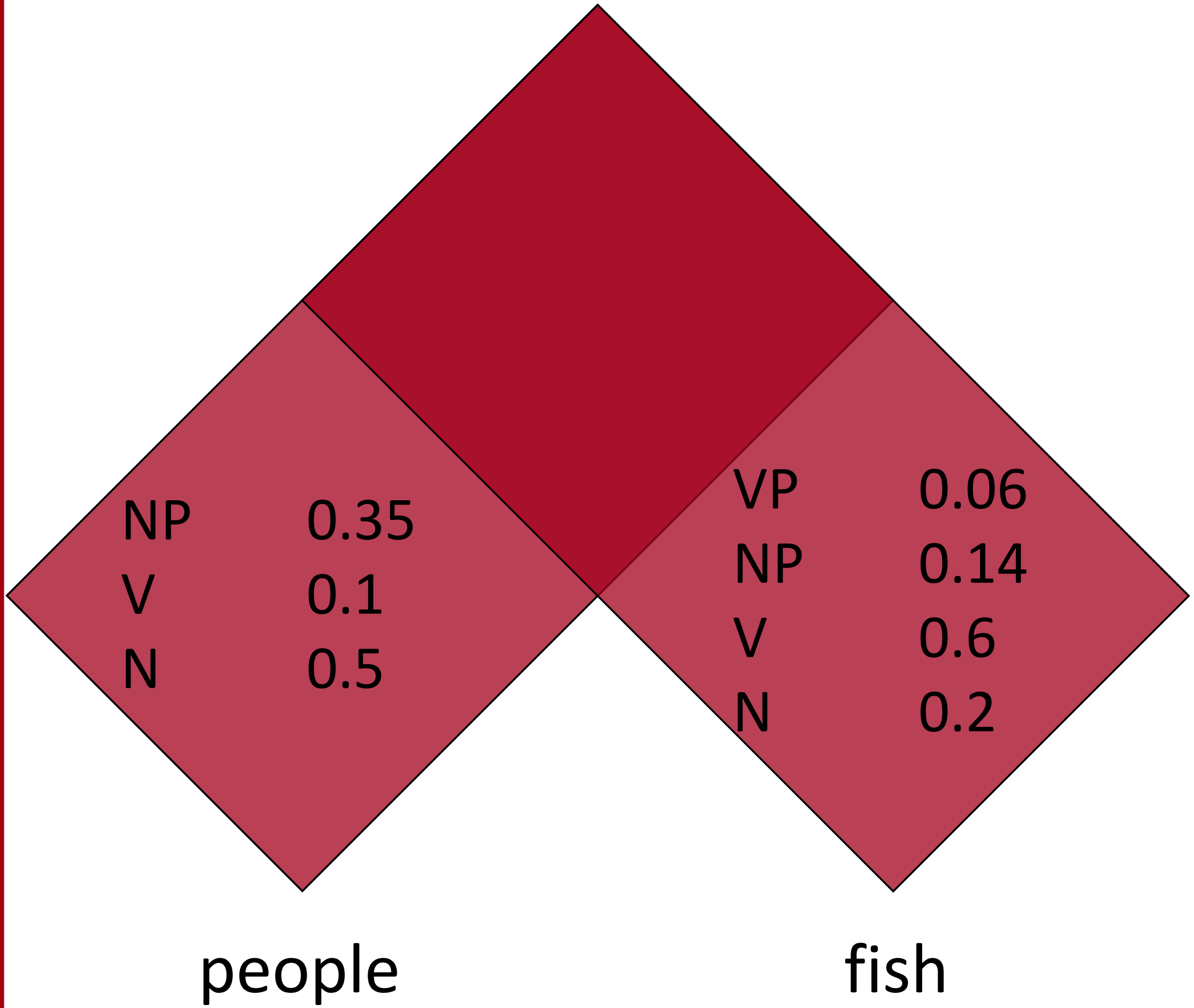
PCFG

Rule	**Prob θ_i**
S → NP VP	θ_0
NP → NP NP	θ_1
…	
N → fish	θ_{42}
N → people	θ_{43}
V → fish	θ_{44}
…	

Cocke-Kasami-Younger (CKY) Constituency Parsing (Parse Triangle/Chart)



Viterbi (Max) Scores



NP 0.35
V 0.1
N 0.5

VP 0.06
NP 0.14
V 0.6
N 0.2

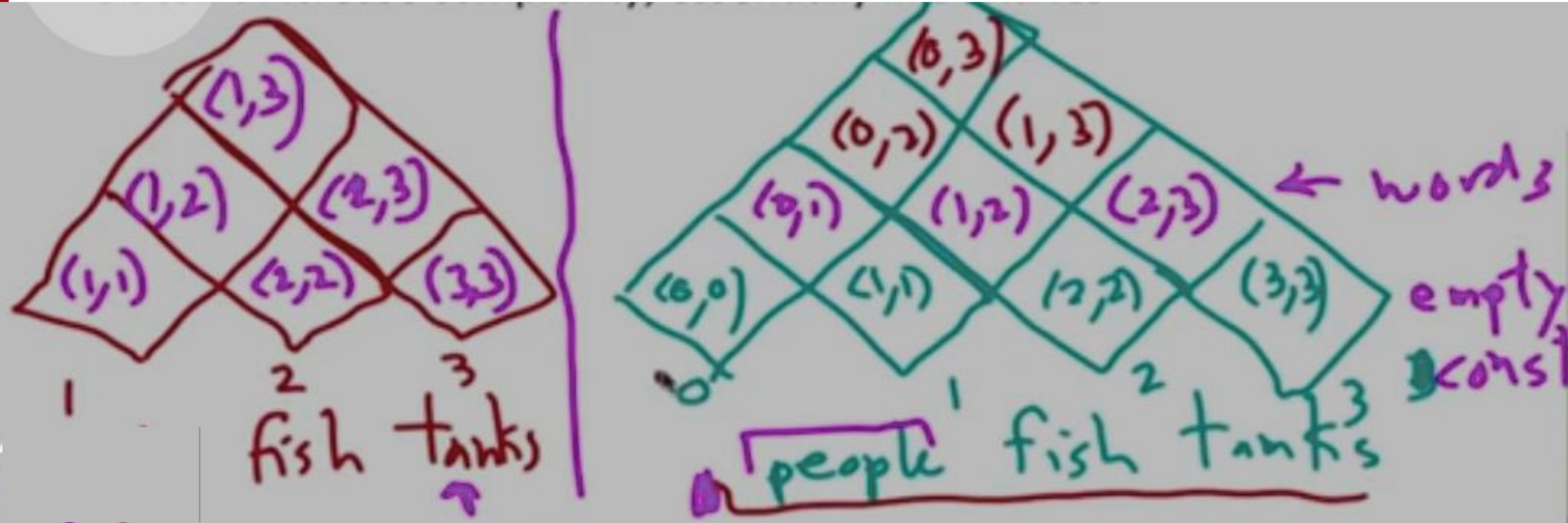
people

fish

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

Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn't increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts
  - Doesn't increase complexity; essentially like unaries



Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts
  - Doesn’t increase complexity; essentially like unaries

- Binarization is *vital*
  - Without binarization, you don’t get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
    - Binarization may be an explicit transformation or implicit in how the parser works (Earley-style dotted rules), but it’s always there.

A Recursive Parser

```
bestScore(X,i,j,s)
 if (j == i)
     return q(X->s[i])
 else
     return max q(X->YZ) *
            k,X->YZ
                  bestScore(Y,i,k,s) *
                  bestScore(Z,k+1,j,s)
```

The CKY algorithm (1960/1965)
… extended to unaries

```
function CKY(words, grammar) returns [most_probable_parse,prob]
  score = new double[#(words)+1][#(words)+1][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]]
//LEXICON
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])
    //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
```

The CKY algorithm (1960/1965) … extended to unaries

```
//build higher order cells
for span = 2 to #(words)
  for begin = 0 to #(words)- span
    end = begin + span
    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)
    //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
return buildTree(score, back)
```

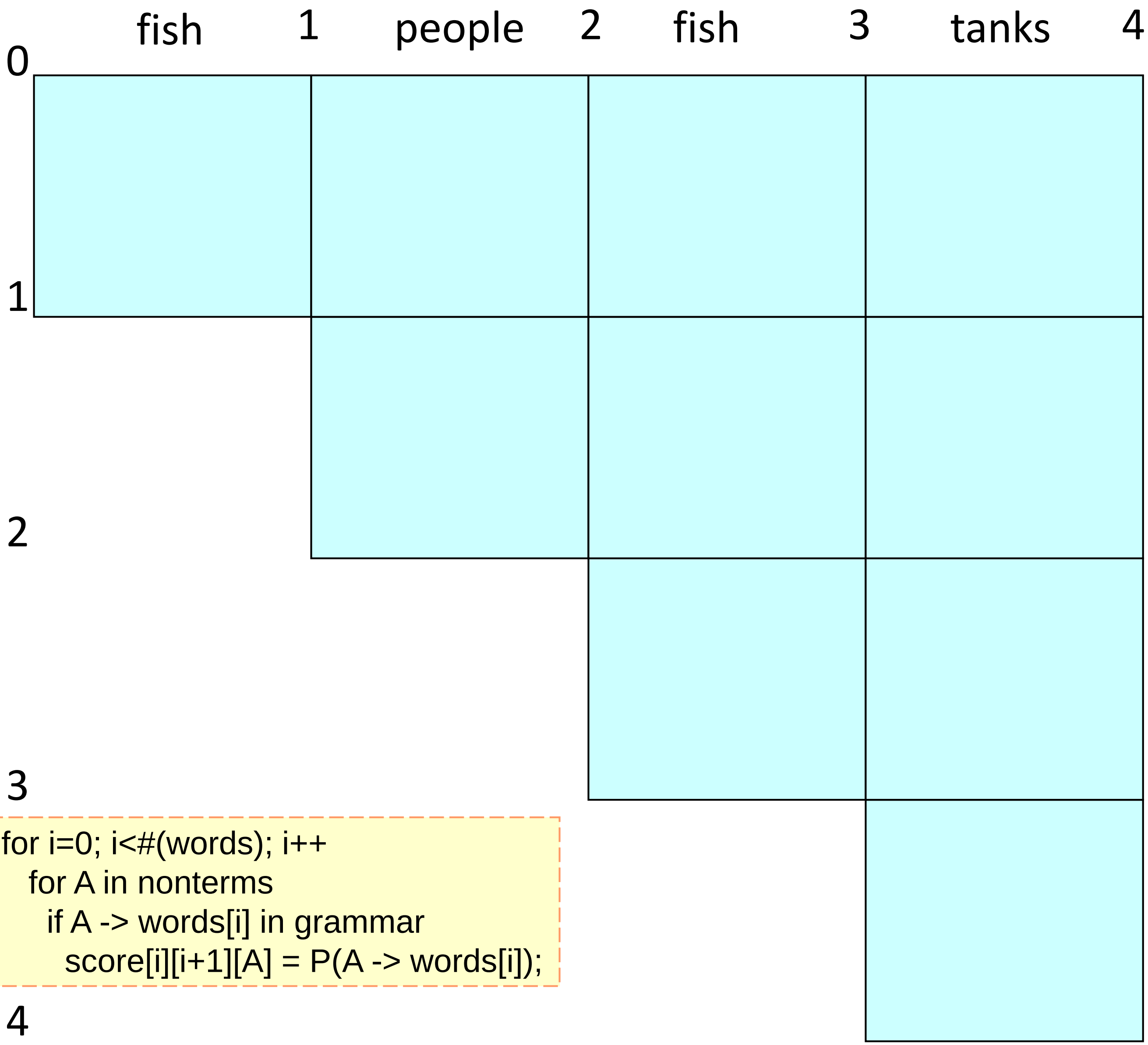
The grammar:
Binary, no epsilons,

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	people	fish	tanks
	1	2	3	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				

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



```
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]);
```

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

```
// 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
```

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			
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
4				

```
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)
```

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 → 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 → NP NP 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 → NP NP 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
4				

```
//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
```

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
1	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		
2		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	
3			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
4				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)
```

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 → NP NP 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 → NP NP 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 → NP NP 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
4				

```
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)
```

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
1	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 → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	
2		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 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
3			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
4				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)
```

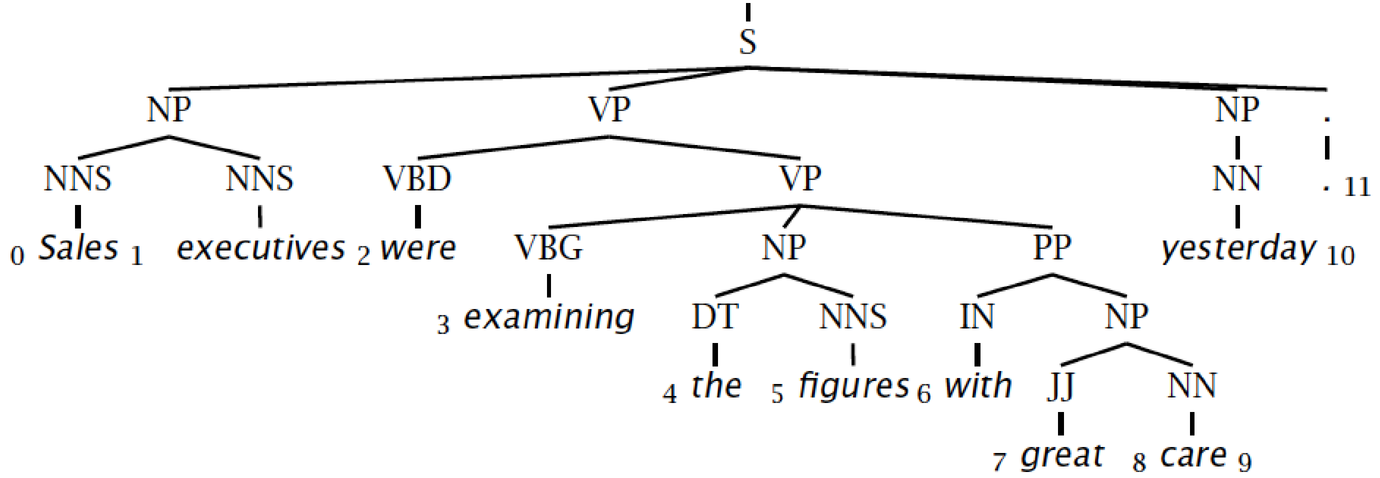
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks* 0.2	
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6
V → *tanks*	0.3
P → *with*	1.0

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 → NP NP 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 → NP NP 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 → NP NP 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
4				

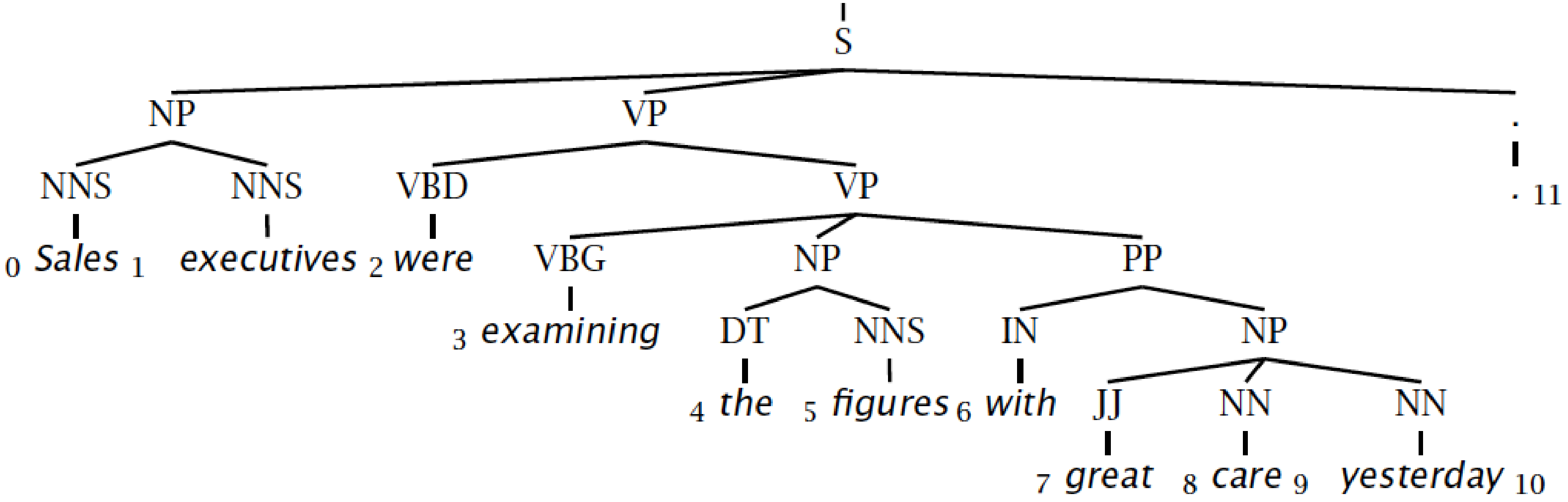
Call buildTree(score, back) to get the best parse

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%

How good are PCFGs?

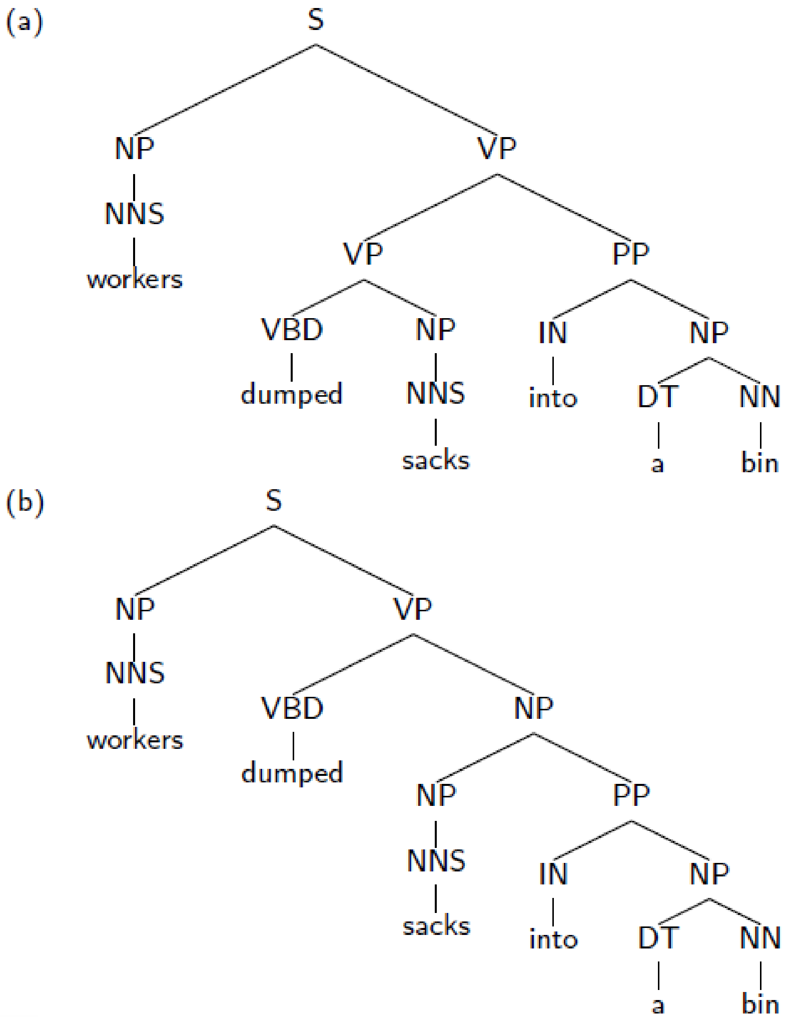
- Penn WSJ parsing accuracy: about 73.7% LP/LR F1
- Robust
  - Usually admit everything, but with low probability
- Partial solution for grammar ambiguity
  - A PCFG gives some idea of the plausibility of a parse
  - But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in the simple case it performs worse than a trigram model
- The problem seems to be that PCFGs lack the lexicalization of a trigram model

Weaknesses of PCFGs

Weaknesses

- Lack of sensitivity to structural frequencies
- Lack of sensitivity to lexical information
- (A word is independent of the rest of the tree given its POS!)

A Case of PP Attachment Ambiguity



(a)

Rules
S → NP VP
NP → NNS
VP → VP PP
VP → VBD NP
NP → NNS
PP → IN NP
NP → DT NN
NNS → workers
VBD → dumped
NNS → sacks
IN → into
DT → a
NN → bin

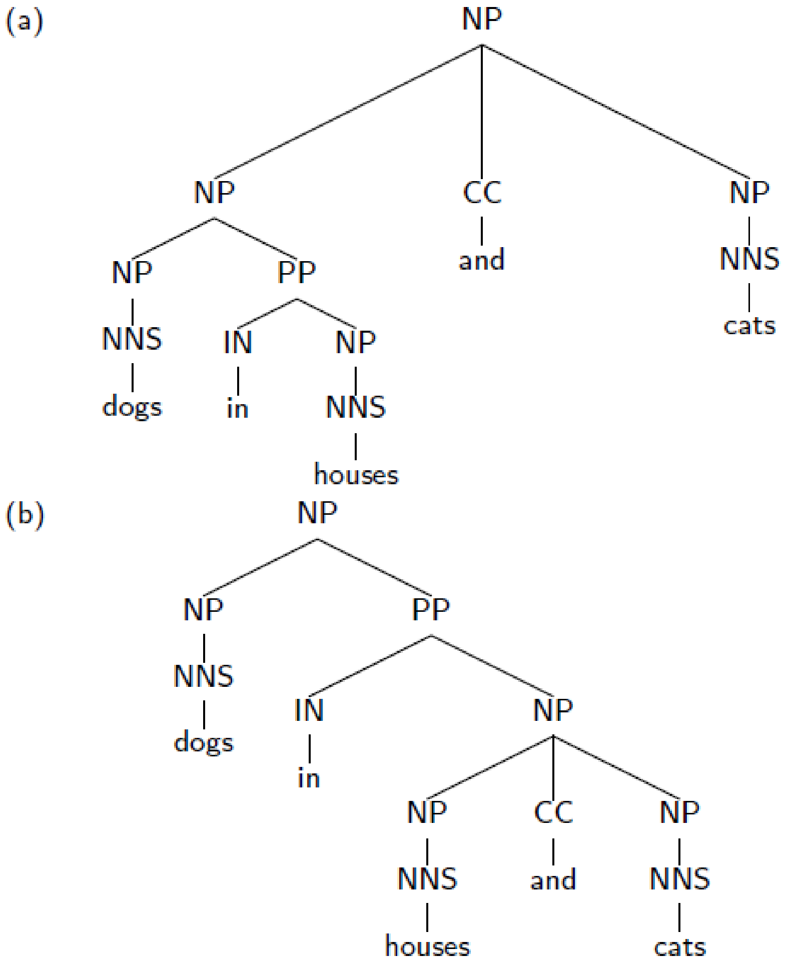
(b)

Rules
S → NP VP
NP → NNS
NP → NP PP
VP → VBD NP
NP → NNS
PP → IN NP
NP → DT NN
NNS → workers
VBD → dumped
NNS → sacks
IN → into
DT → a
NN → bin

If $q(\text{NP} \rightarrow \text{NP PP}) > q(\text{VP} \rightarrow \text{VP PP})$ then (b) is more probable, else (a) is more probable.

Attachment decision is completely independent of the words

A Case of Coordination Ambiguity



(a)

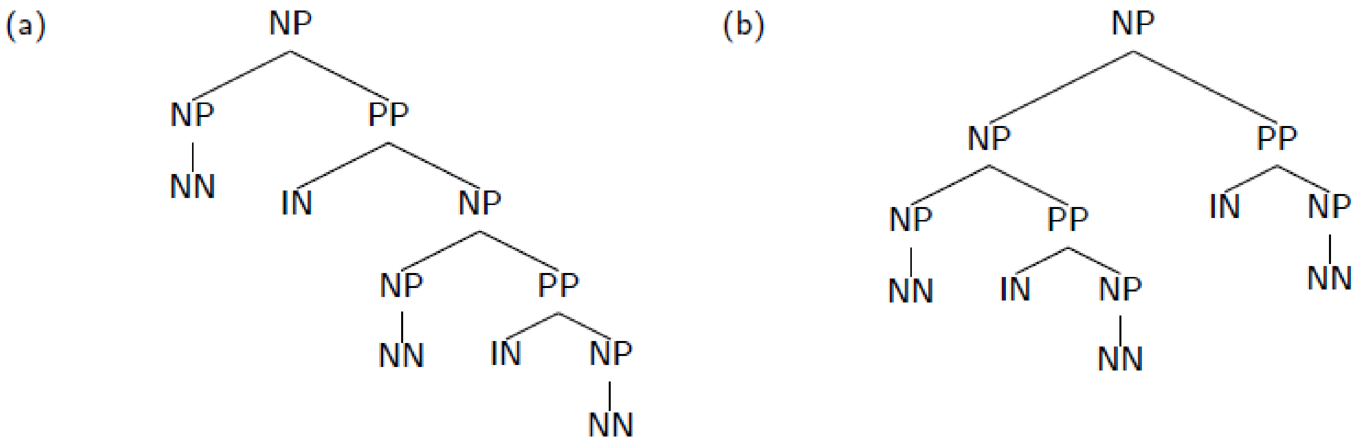
Rules
NP → NP CC NP
NP → NP PP
NP → NNS
PP → IN NP
NP → NNS
NP → NNS
NNS → dogs
IN → in
NNS → houses
CC → and
NNS → cats

(b)

Rules
NP → NP CC NP
NP → NP PP
NP → NNS
PP → IN NP
NP → NNS
NP → NNS
NNS → dogs
IN → in
NNS → houses
CC → and
NNS → cats

Here the two parses have identical rules, and therefore have identical probability under any assignment of PCFG rule probabilities

Structural Preferences: Close Attachment



- Example: president of a company in Africa
- Both parses have the same rules, therefore receive same probability under a PCFG
- "Close attachment" (structure (a)) is twice as likely in Wall Street Journal text.

Structural Preferences: Close Attachment

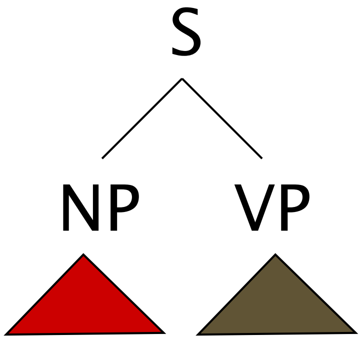
- **Example:** John was believed to have been shot by Bill
- Low attachment analysis (Bill does the shooting) contains same rules as high attachment analysis (Bill does the believing)
  - Two analyses receive the same probability

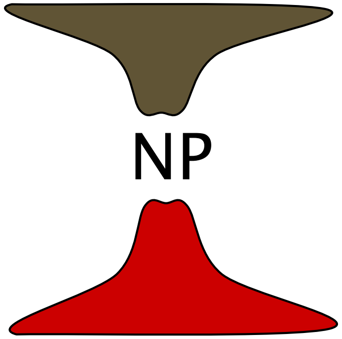
PCFGs and Independence

- The symbols in a PCFG define independence assumptions:

S → NP VP

NP → DT NN

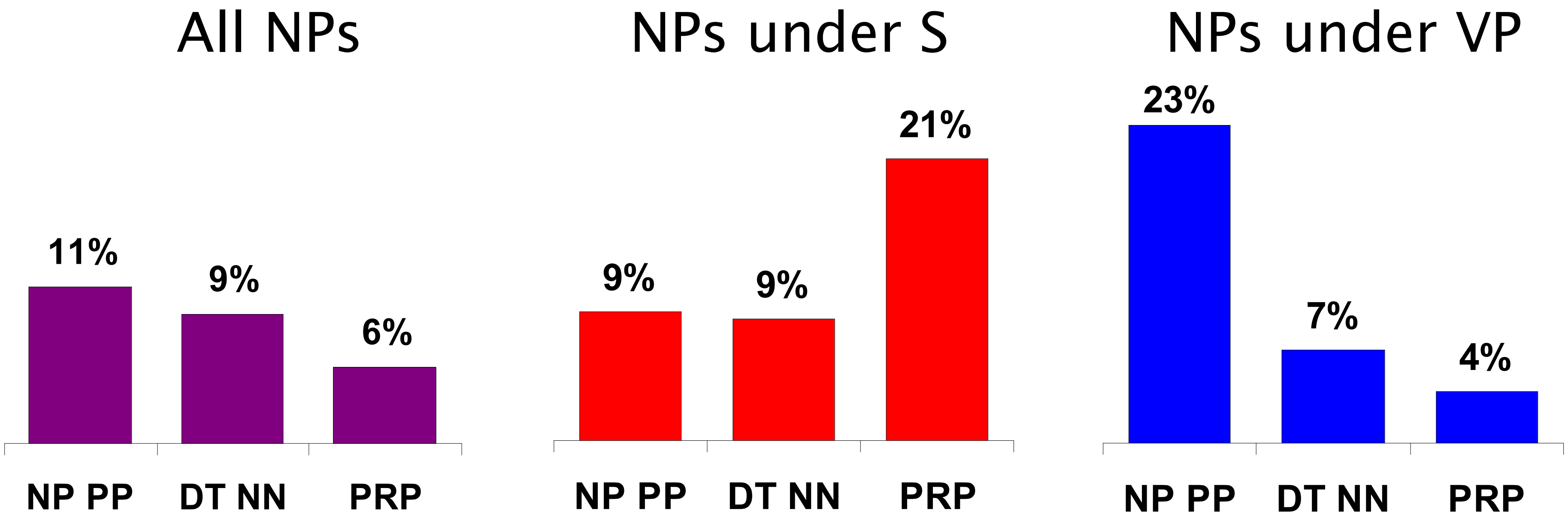




- At any node, the material inside that node is independent of the material outside that node, given the label of that node
- Any information that statistically connects behavior inside and outside a node must flow through that node's label

Non-Independence I

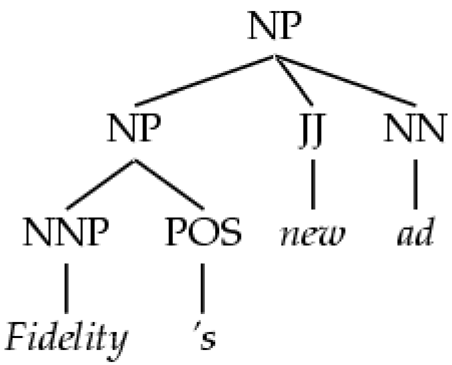
- The independence assumptions of a PCFG are often too strong

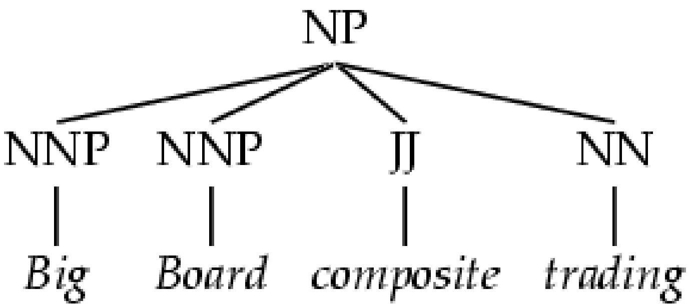


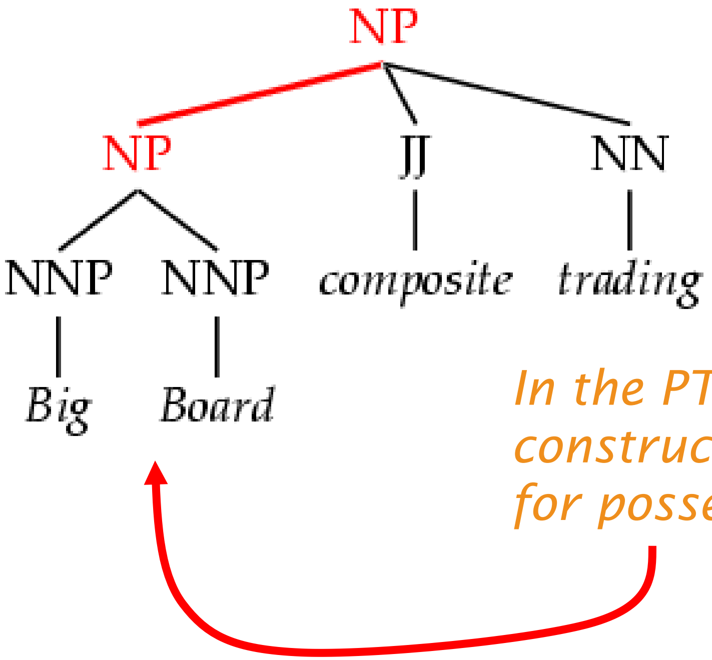
- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects)

Non-Independence II

- Symptoms of overly strong assumptions:
  - Rewrites get used where they don't belong





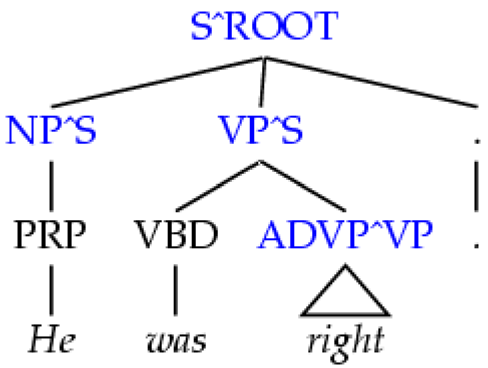


In the PTB, this construction is for possessives

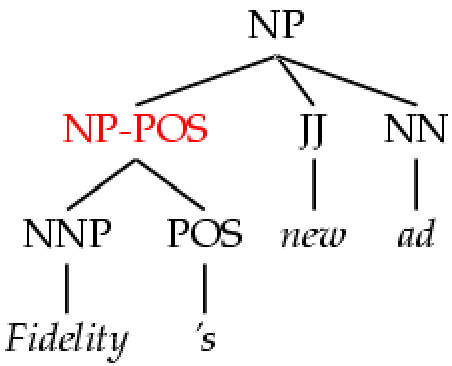
Refining the Grammar Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols, by state splitting:

Parent annotation
[Johnson 98]



Marking
possessive NPs

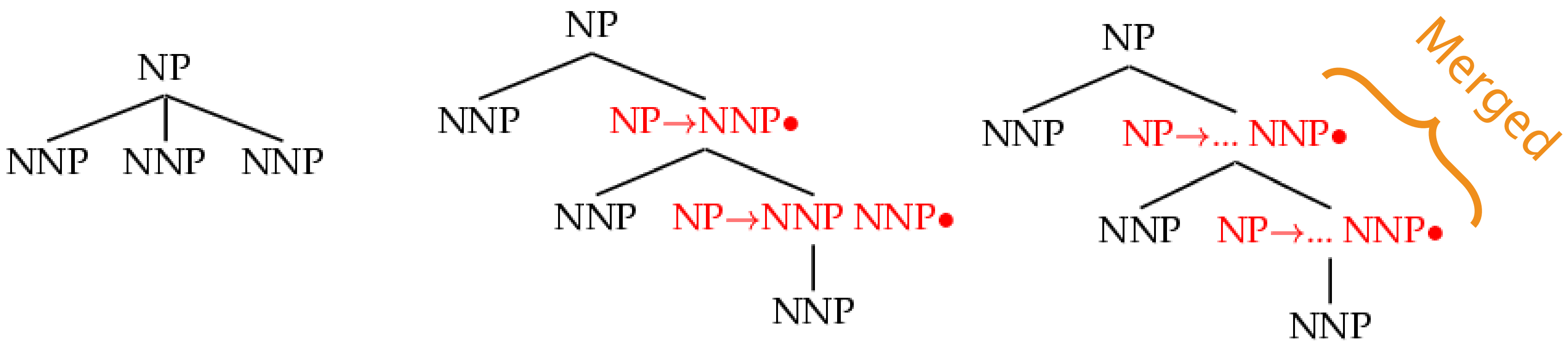


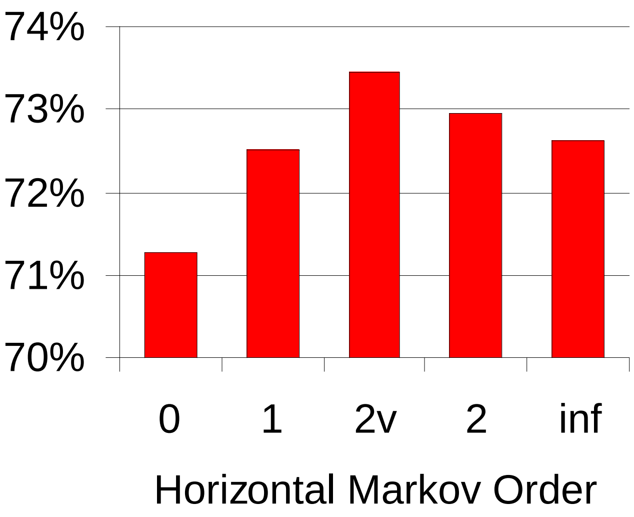
- Too much state-splitting → sparseness (no smoothing used!)
- What are the most useful features to encode?

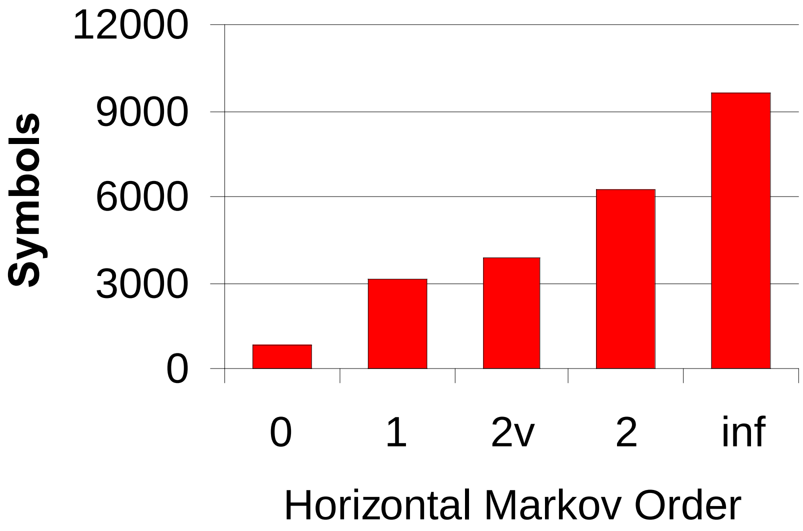
Linguistics in Unlexicalized Parsing

Horizontal Markovization

- Horizontal Markovization: Merges States



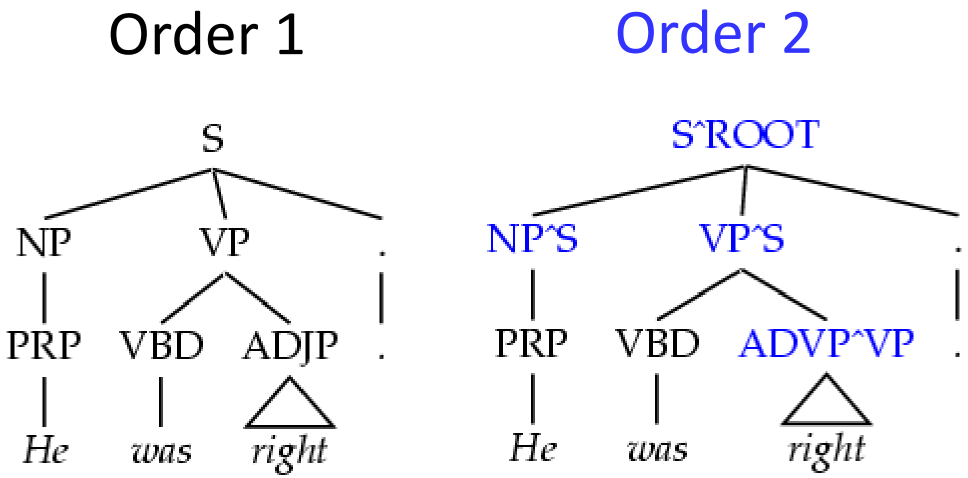


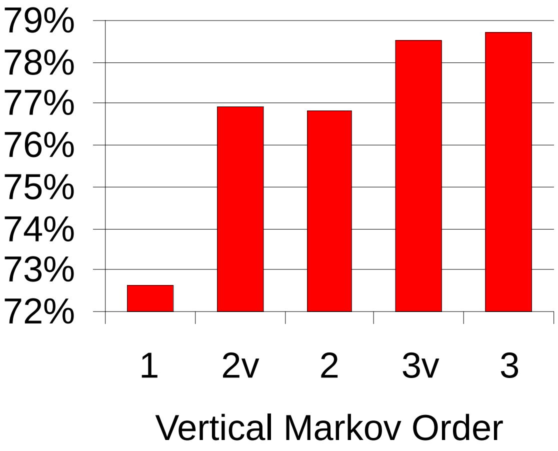


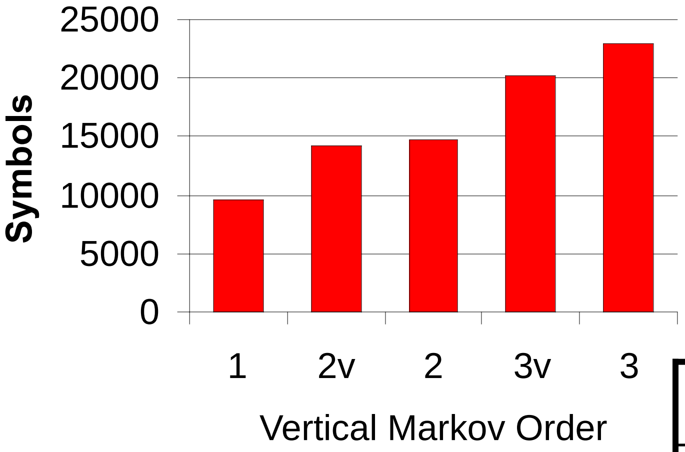
Vertical Markovization

- Vertical Markov order: rewrites depend on past *k* ancestor nodes.

  (i.e., parent annotation)



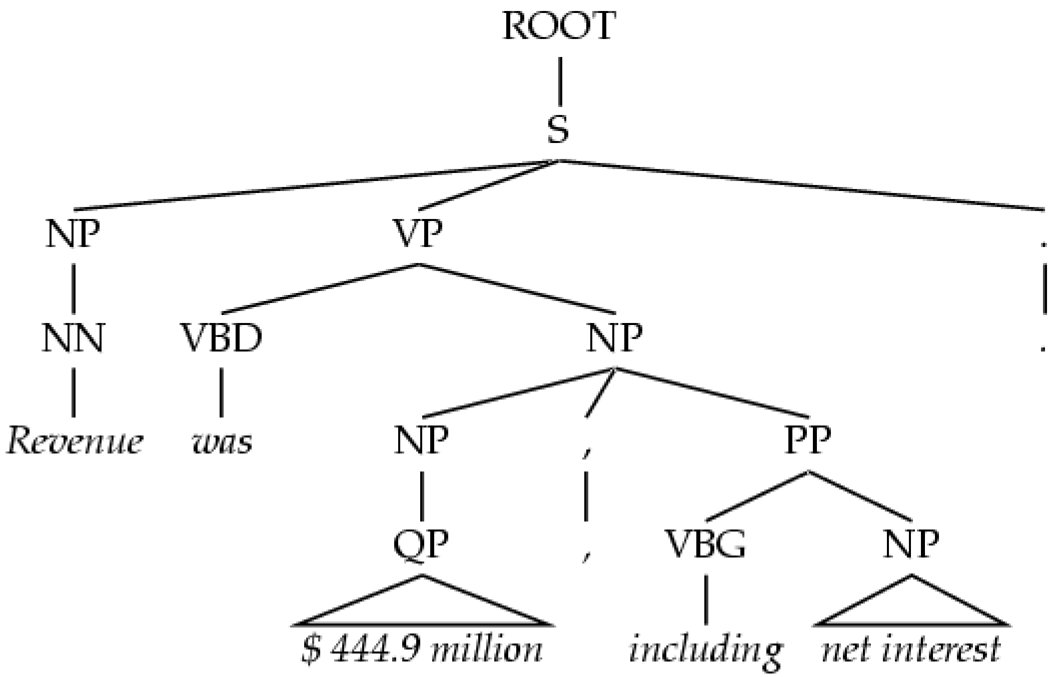




Model	F1	Size
v=h=2v	77.8	7.5K

Unary Splits

- Problem: unary rewrites are used to transmute categories so a high-probability rule can be used.

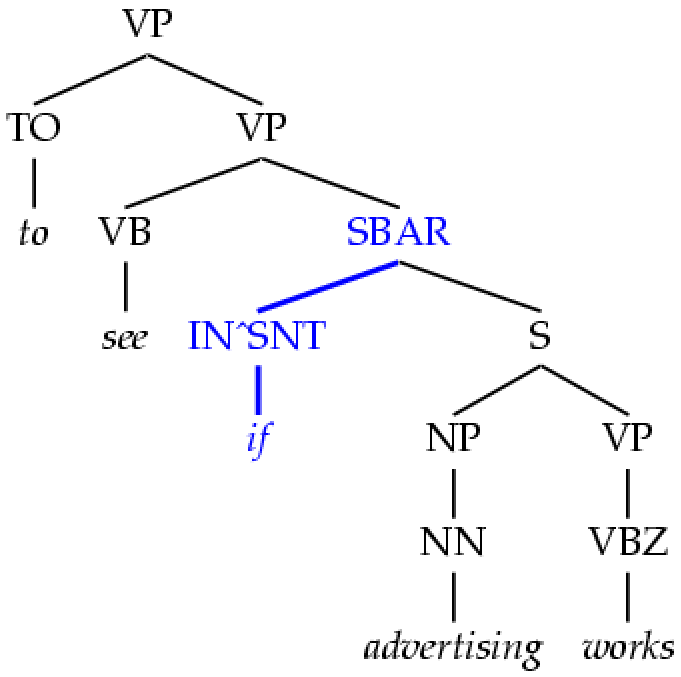


- Solution: Mark unary rewrite sites with -U

Annotation	F1	Size
Base	77.8	7.5K
UNARY	78.3	8.0K

Tag Splits

- Problem: Treebank tags are too coarse.
- Example: SBAR sentential complementizers (*that, whether, if)*, subordinating conjunctions (*while, after)*, and true prepositions (*in, of, to)* are all tagged IN.
- Partial Solution:
  - Subdivide the IN tag.



Annotation	F1	Size
Previous	78.3	8.0K
SPLIT-IN	80.3	8.1K

Other Tag Splits

- UNARY-DT: mark demonstratives as DT^U (“the X” vs. “those”)
- UNARY-RB: mark phrasal adverbs as RB^U (“quickly” vs. “very”)
- TAG-PA: mark tags with non-canonical parents (“not” is an RB^VP)
- SPLIT-AUX: mark auxiliary verbs with –AUX [cf. Charniak 97]
- SPLIT-CC: separate “but” and “&” from other conjunctions
- SPLIT-%: “%” gets its own tag.

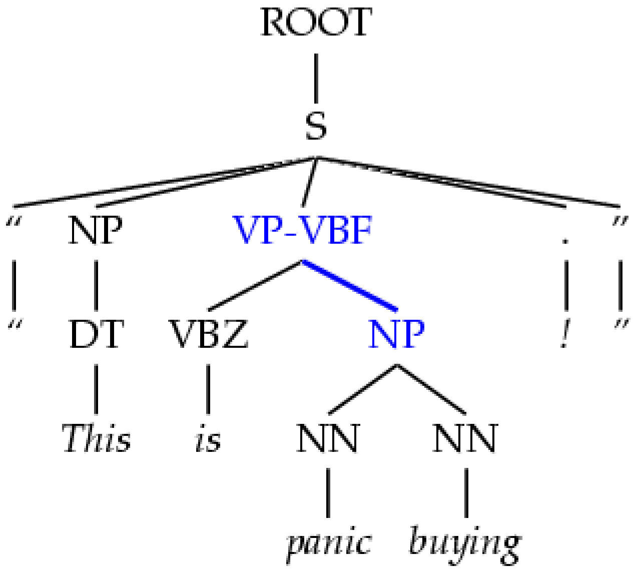
F1	Size
80.4	8.1K
80.5	8.1K
81.2	8.5K
81.6	9.0K
81.7	9.1K
81.8	9.3K

Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.

- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!

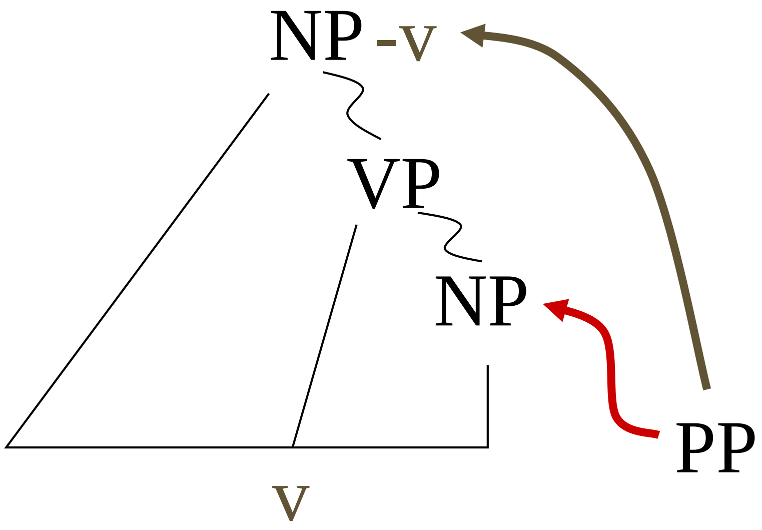
- Solution: annotate future elements into nodes.



Annotation	F1	Size
tag splits	82.3	9.7K
POSS-NP	83.1	9.8K
SPLIT-VP	85.7	10.5K

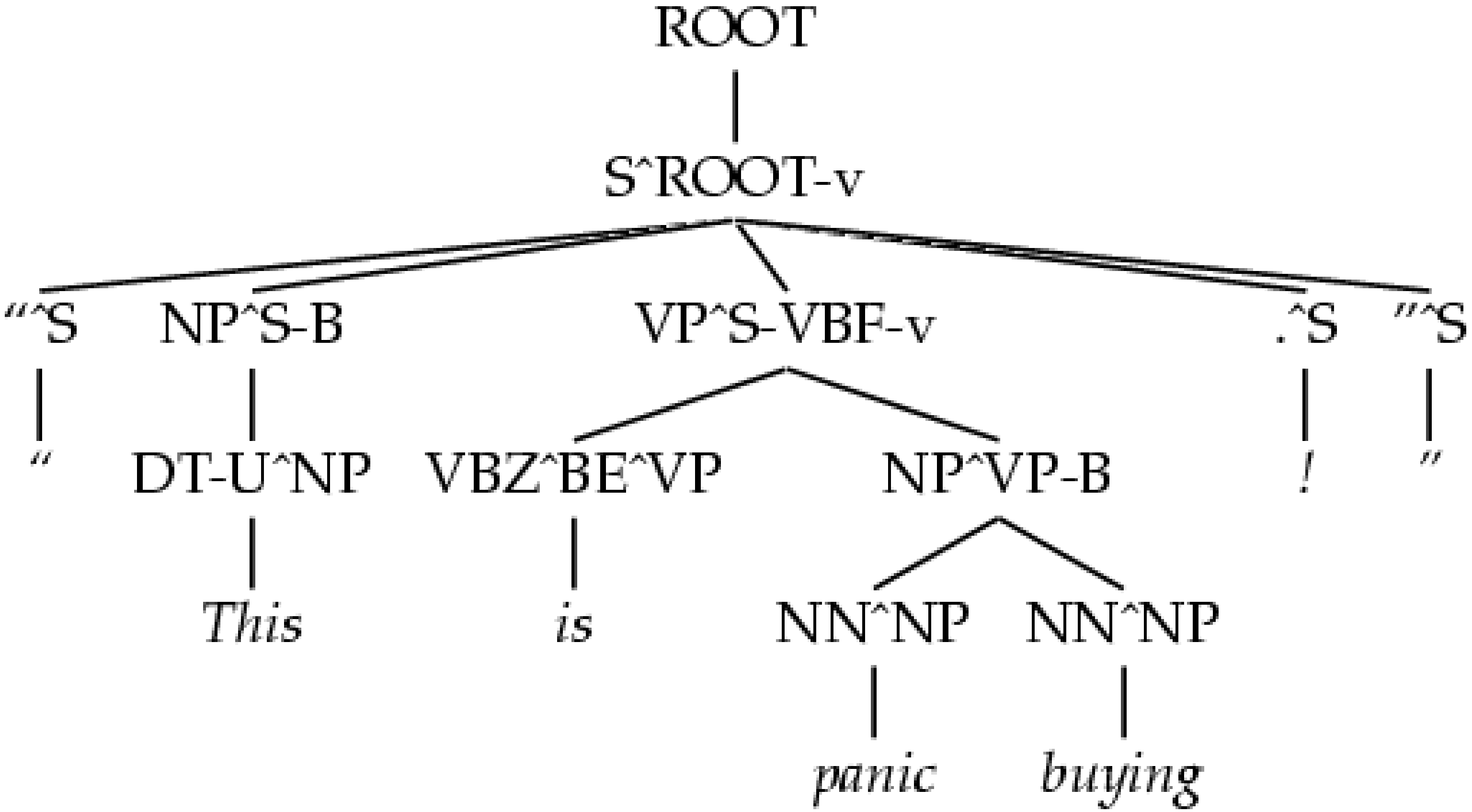
Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.
- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
    - Base NPs [cf. Collins 99]
    - Right-recursive NPs



Annotation	F1	Size
Previous	85.7	10.5K
BASE-NP	86.0	11.7K
DOMINATES-V	86.9	14.1K
RIGHT-REC-NP	87.0	15.2K

A Fully Annotated Tree



Final Test Set Results

Parser	LP	LR	**F1**
Magerman 95	84.9	84.6	**84.7**
Collins 96	86.3	85.8	**86.0**
Klein & Manning 03	86.9	85.7	**86.3**
Charniak 97	87.4	87.5	**87.4**
Collins 99	88.7	88.6	**88.6**

- Beats “first generation” lexicalized parsers

Lexicalised PCFGs

Heads in Context Free Rules

Add annotations specifying the “head” of each rule:

S	⇒	NP	VP
VP	⇒	Vi	
VP	⇒	Vt	NP
VP	⇒	VP	PP
NP	⇒	DT	NN
NP	⇒	NP	PP
PP	⇒	IN	NP

Vi	⇒	sleeps
Vt	⇒	saw
NN	⇒	man
NN	⇒	woman
NN	⇒	telescope
DT	⇒	the
IN	⇒	with
IN	⇒	in

Heads

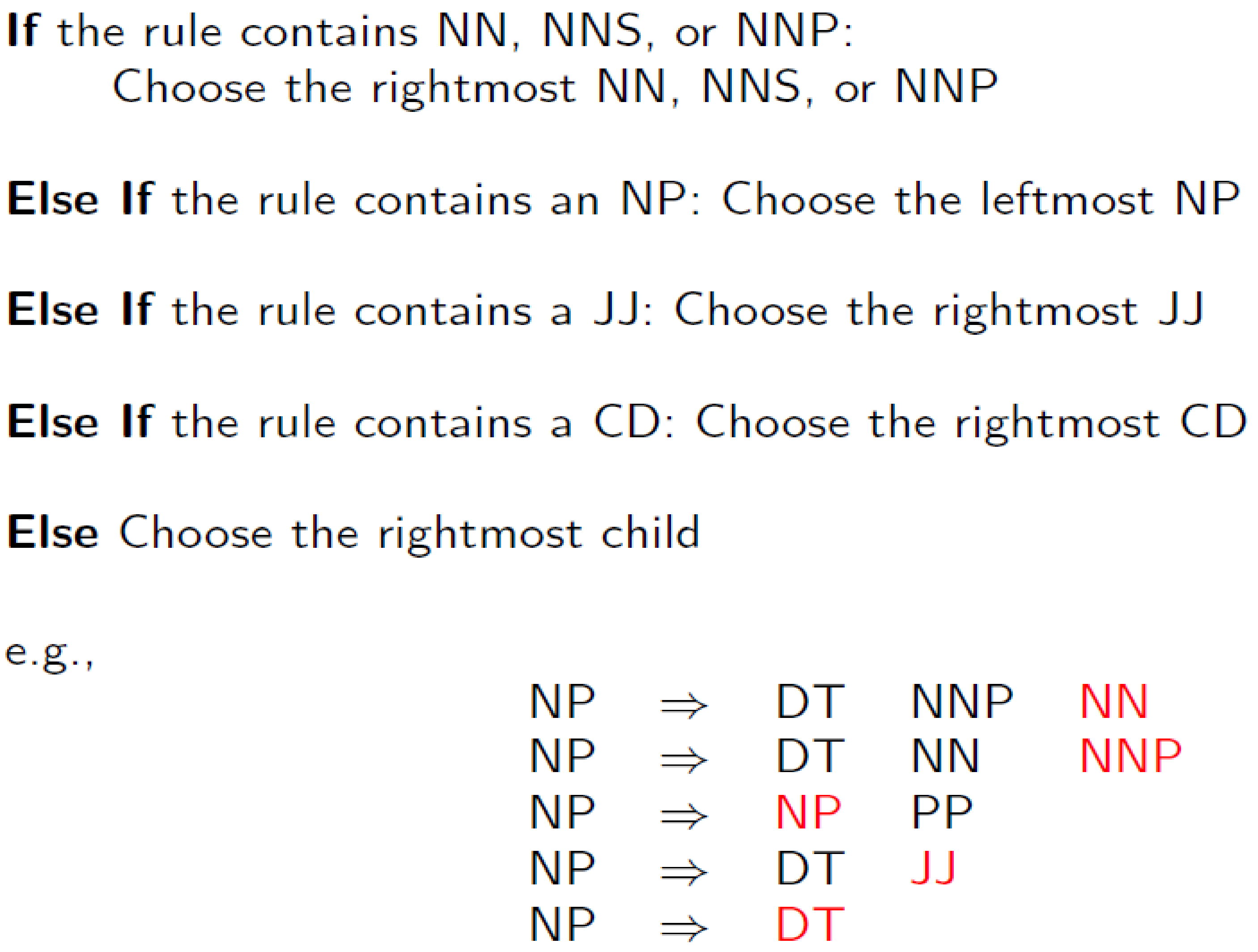
- Each context-free rule has one "special" child that is the head of the rule. e.g.,

S	⇒	NP	VP		(VP is the head)
VP	⇒	Vt	NP		(Vt is the head)
NP	⇒	DT	NN	NN	(NN is the head)

- A core idea in syntax
(e.g., see X-bar Theory, Head-Driven Phrase Structure Grammar)

- Some intuitions:
  - The central sub-constituent of each rule.
  - The semantic predicate in each rule.

Rules to Recover Heads: An Example for NPs



If the rule contains NN, NNS, or NNP:
    Choose the rightmost NN, NNS, or NNP

Else If the rule contains an NP: Choose the leftmost NP

Else If the rule contains a JJ: Choose the rightmost JJ

Else If the rule contains a CD: Choose the rightmost CD

Else Choose the rightmost child

e.g.,

$$\begin{array}{lllll} NP & \Rightarrow & DT & NNP & NN \\ NP & \Rightarrow & DT & NN & NNP \\ NP & \Rightarrow & NP & PP & \\ NP & \Rightarrow & DT & JJ & \\ NP & \Rightarrow & DT & & \end{array}$$

Rules to Recover Heads: An Example for VPs

If the rule contains Vi or Vt: Choose the leftmost Vi or Vt

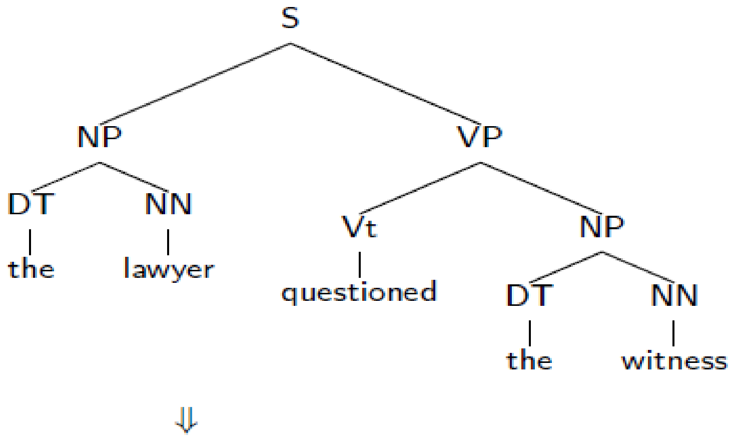
Else If the rule contains an VP: Choose the leftmost VP

Else Choose the leftmost child

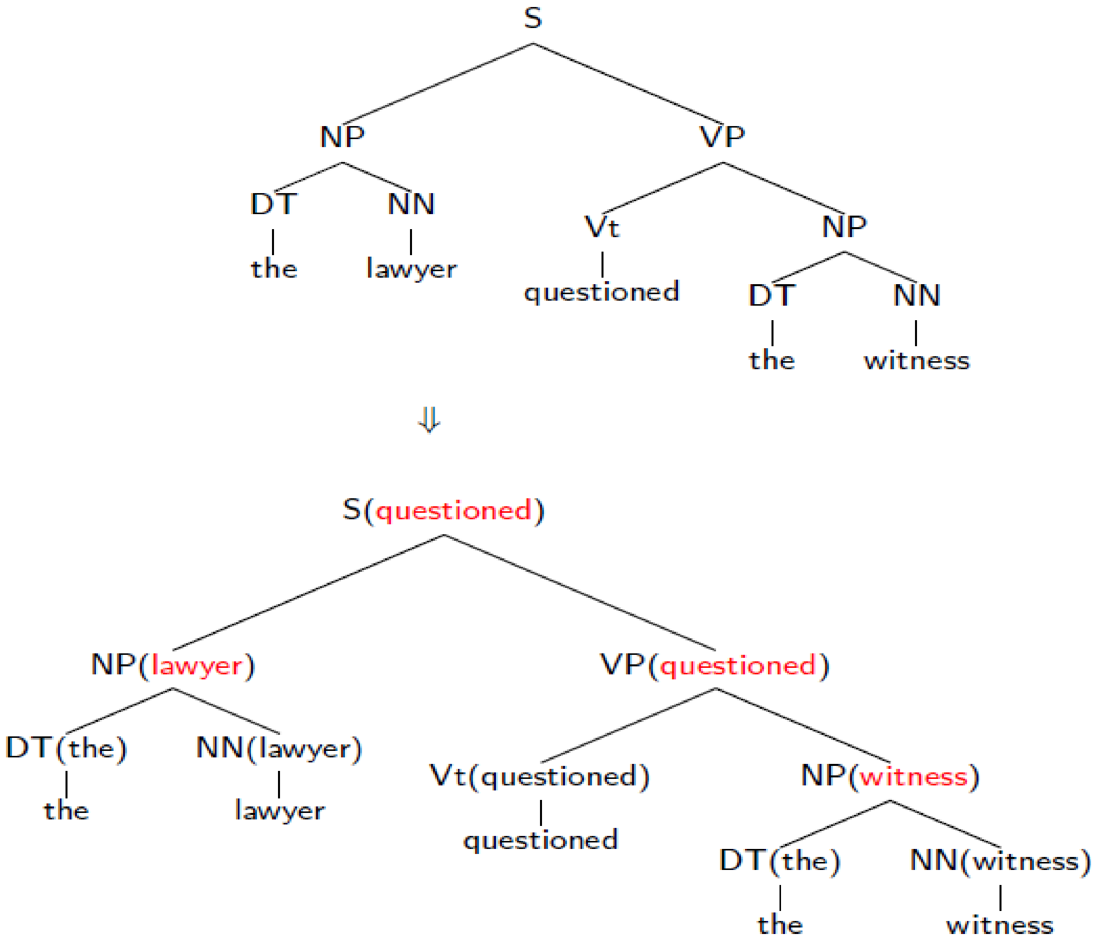
e.g.,

$$\begin{array}{llll} \text{VP} & \Rightarrow & \text{Vt} & \text{NP} \\ \text{VP} & \Rightarrow & \text{VP} & \text{PP} \end{array}$$

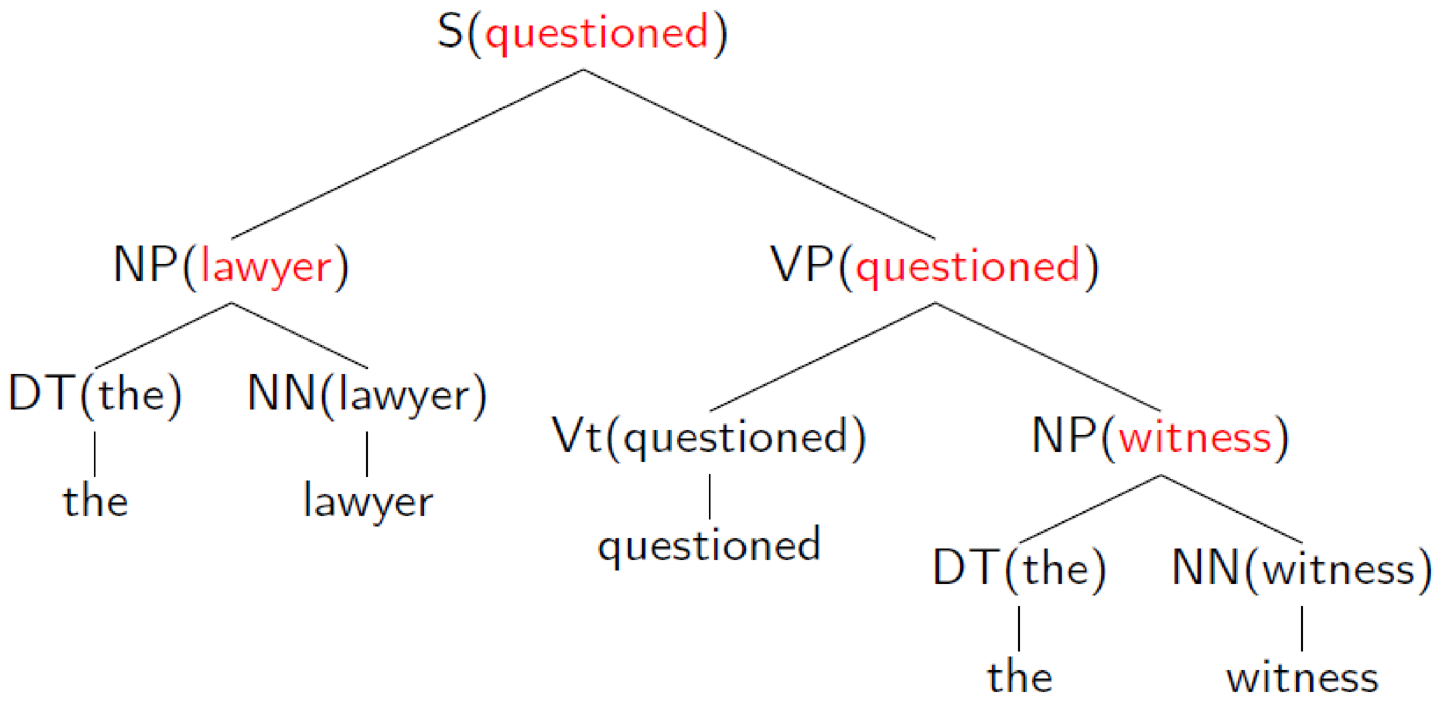
Adding Headwords to Trees



Adding Headwords to Trees



Adding Headwords to Trees



- A constituent receives its headword from its **head child**.

S	⇒	NP	VP		(S receives headword from VP)
VP	⇒	Vt	NP		(VP receives headword from Vt)
NP	⇒	DT		NN	(NP receives headword from NN)

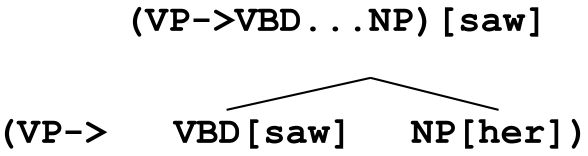
Lexicalized CFGs in Chomsky Normal Form

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules which take one of three forms:
  - $X(h) \rightarrow_1 Y_1(h)\ Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow_2 Y_1(w)\ Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
- $S \in N$ is a distinguished start symbol

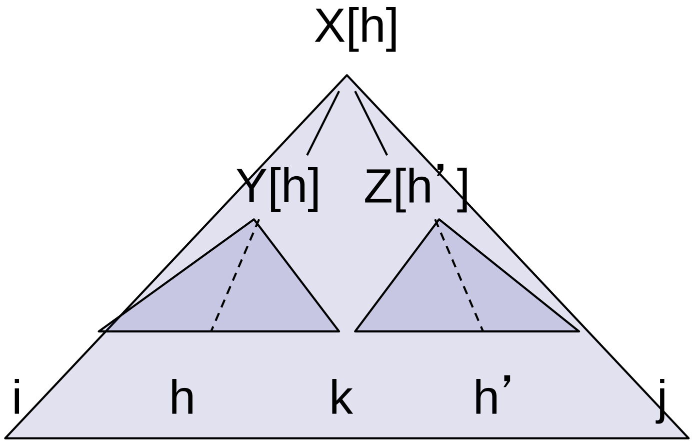
Example

$$
\begin{array}{llll}
\text{S(saw)} & \rightarrow_2 & \text{NP(man)} & \text{VP(saw)} \\
\text{VP(saw)} & \rightarrow_1 & \text{Vt(saw)} & \text{NP(dog)} \\
\text{NP(man)} & \rightarrow_2 & \text{DT(the)} & \text{NN(man)} \\
\text{NP(dog)} & \rightarrow_2 & \text{DT(the)} & \text{NN(dog)} \\
\text{Vt(saw)} & \rightarrow & \text{saw} & \\
\text{DT(the)} & \rightarrow & \text{the} & \\
\text{NN(man)} & \rightarrow & \text{man} & \\
\text{NN(dog)} & \rightarrow & \text{dog} &
\end{array}
$$

Lexicalized CKY


(VP->VBD...NP)[saw]

(VP-> VBD[saw] NP[her])



```
bestScore(X,i,j,h)
  if (j = i)
    return score(X,s[i])
  else
    return
            max     score(X[h]->Y[h]Z[w]) *
          k,h,w
                    bestScore(Y,i,k,h) *
          X->YZ
                    bestScore(Z,k+1,j,w)
            max     score(X[h]->Y[w]Z[h]) *
          k,h,w
                    bestScore(Y,i,k,w) *
          X->YZ
                    bestScore(Z,k+1,j,h)
```

Parsing with Lexicalized CFGs

- The new form of grammar looks just like a Chomsky normal form CFG, but with potentially $O(|\Sigma|^2 \times |N|^3)$ possible rules.

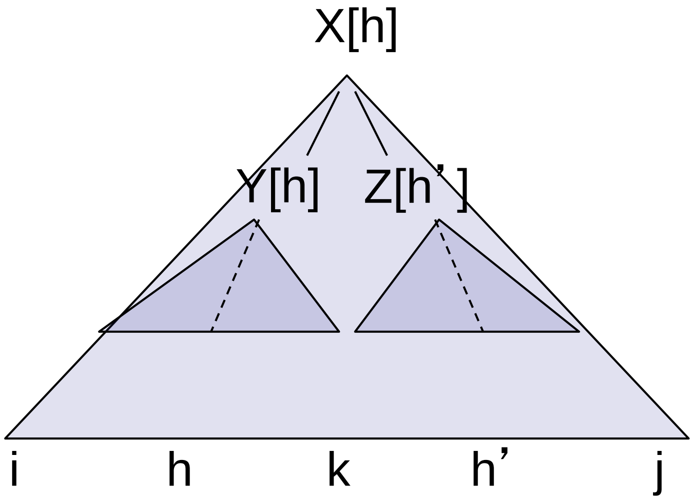
- Naively, parsing an n word sentence using the dynamic programming algorithm will take $O(n^3|\Sigma|^2|N|^3)$ time. But $|\Sigma|$ can be huge!!

- Crucial observation: at most $O(n^2 \times |N|^3)$ rules can be applicable to a given sentence $w_1, w_2, \ldots w_n$ of length n. This is because any rules which contain a lexical item that is not one of $w_1 \ldots w_n$, can be safely discarded.

- The result: we can parse in $O(n^5|N|^3)$ time.

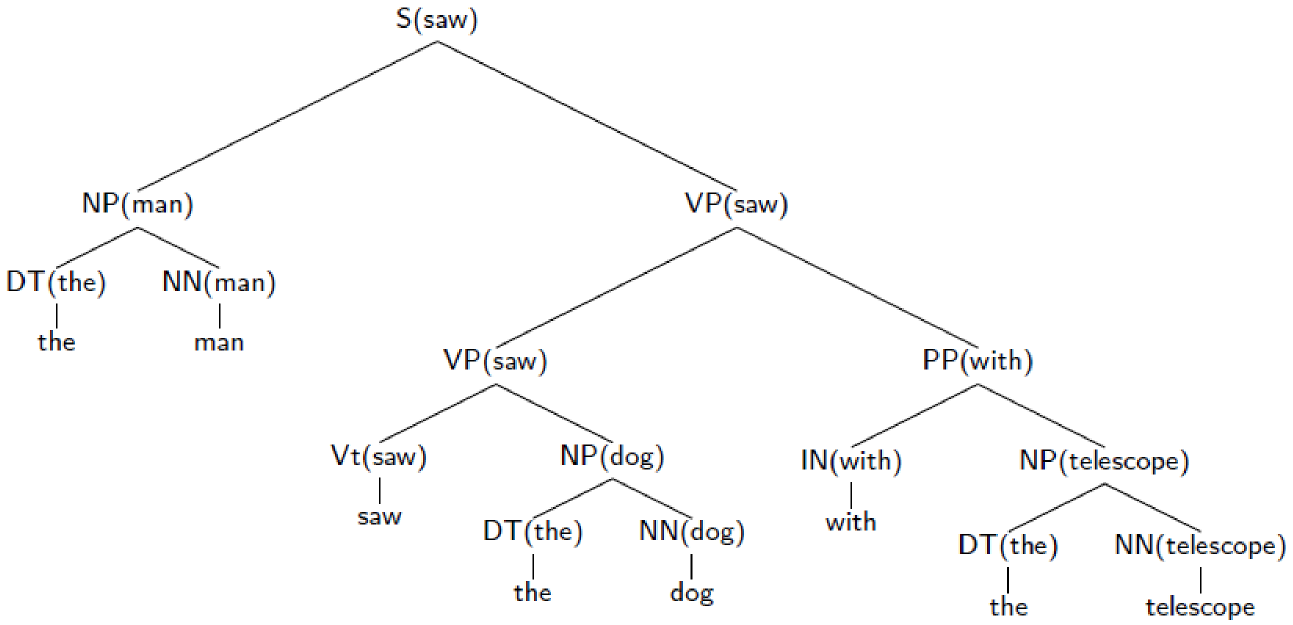
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ CKY
  - Remember only a few hypotheses for each span <i,j>.
  - If we keep K hypotheses at each span, then we do at most $O(nK^2)$ work per span (why?)
  - Keeps things more or less cubic



- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)

Parameter Estimation



$$
\begin{aligned}
\mathsf{p(t)} = {} & q(\mathsf{S(saw)} \rightarrow_2 \mathsf{NP(man)\ VP(saw)}) \\
& \times q(\mathsf{NP(man)} \rightarrow_2 \mathsf{DT(the)\ NN(man)}) \\
& \times q(\mathsf{VP(saw)} \rightarrow_1 \mathsf{VP(saw)\ PP(with)}) \\
& \times q(\mathsf{VP(saw)} \rightarrow_1 \mathsf{Vt(saw)\ NP(dog)}) \\
& \times q(\mathsf{PP(with)} \rightarrow_1 \mathsf{IN(with)\ NP(telescope)}) \\
& \times \ldots
\end{aligned}
$$

A Model from Charniak (1997)

- An example parameter in a Lexicalized PCFG:

$$q(\mathsf{S(saw)} \rightarrow_2 \mathsf{NP(man)\ VP(saw)})$$

- First step: decompose this parameter into a product of two parameters

$$\begin{aligned} & q(\mathsf{S(saw)} \rightarrow_2 \mathsf{NP(man)\ VP(saw)}) \\ = \ & q(\mathsf{S} \rightarrow_2 \mathsf{NP\ VP} | \mathsf{S,\ saw}) \times q(\mathsf{man} | \mathsf{S} \rightarrow_2 \mathsf{NP\ VP,\ saw}) \end{aligned}$$

A Model from Charniak (1997)

$$\begin{aligned} & q(\text{S(saw)} \rightarrow_2 \text{NP(man) VP(saw)}) \\ = \; & q(\text{S} \rightarrow_2 \text{NP VP}|\text{S, saw}) \times q(\text{man}|\text{S} \rightarrow_2 \text{NP VP, saw}) \end{aligned}$$

- Second step: use smoothed estimation for the two parameter estimates

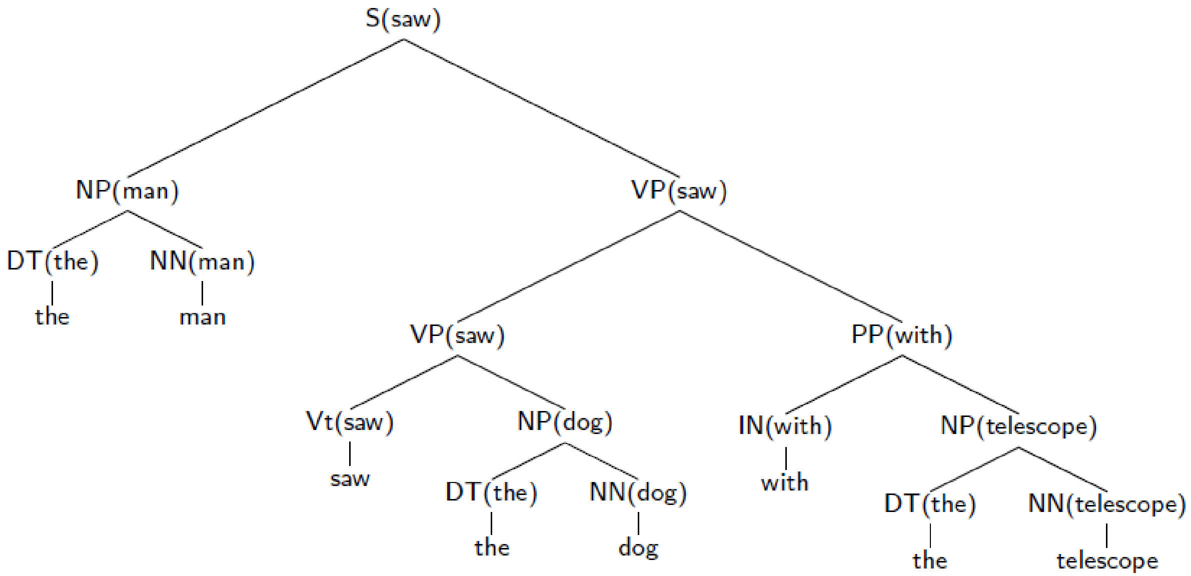
$$\begin{aligned} & q(\text{S} \rightarrow_2 \text{NP VP}|\text{S, saw}) \\ = \; & \lambda_1 \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S, saw}) + \lambda_2 \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S}) \end{aligned}$$

$$\begin{aligned} & q(\text{man}|\text{S} \rightarrow_2 \text{NP VP, saw}) \\ = \; & \lambda_3 \times q_{ML}(\text{man}|\text{S} \rightarrow_2 \text{NP VP, saw}) + \lambda_4 \times q_{ML}(\text{man}|\text{S} \rightarrow_2 \text{NP VP}) \\ & + \lambda_5 \times q_{ML}(\text{man}|\text{NP}) \end{aligned}$$

Final Test Set Results

Parser	LP	LR	**F1**
Magerman 95	84.9	84.6	**84.7**
Collins 96	86.3	85.8	**86.0**
Klein & Manning 03	86.9	85.7	**86.3**
Charniak 97	87.4	87.5	**87.4**
Collins 99	88.7	88.6	**88.6**

Analysis/Evaluation (Method 2)



$\langle\ \mathsf{ROOT}_0,\quad \mathsf{saw}_3,\quad \mathsf{ROOT}\ \rangle$
$\langle\ \mathsf{saw}_3,\quad \mathsf{man}_2,\quad \mathsf{S} \rightarrow_2 \mathsf{NP\ VP}\ \rangle$
$\langle\ \mathsf{man}_2,\quad \mathsf{the}_1,\quad \mathsf{NP} \rightarrow_2 \mathsf{DT\ NN}\ \rangle$
$\langle\ \mathsf{saw}_3,\quad \mathsf{with}_6,\quad \mathsf{VP} \rightarrow_1 \mathsf{VP\ PP}\ \rangle$
$\langle\ \mathsf{saw}_3,\quad \mathsf{dog}_5,\quad \mathsf{VP} \rightarrow_1 \mathsf{Vt\ NP}\ \rangle$
$\langle\ \mathsf{dog}_5,\quad \mathsf{the}_4,\quad \mathsf{NP} \rightarrow_2 \mathsf{DT\ NN}\ \rangle$
$\langle\ \mathsf{with}_6,\quad \mathsf{telescope}_8,\quad \mathsf{PP} \rightarrow_1 \mathsf{IN\ NP}\ \rangle$
$\langle\ \mathsf{telescope}_8,\quad \mathsf{the}_7,\quad \mathsf{NP} \rightarrow_2 \mathsf{DT\ NN}\ \rangle$

Dependency Accuracies

- All parses for a sentence with n words have n dependencies
  Report a single figure, dependency accuracy

- Results from Collins, 2003: 88.3% dependency accuracy

- Can calculate precision/recall on particular dependency types
  e.g., look at all subject/verb dependencies ⇒
  all dependencies with label S $\rightarrow_2$ NP VP

  $$\text{Recall} = \frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in gold standard}}$$

  $$\text{Precision} = \frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in parser's output}}$$

Strengths and Weaknesses of PCFG Parsers

(Numbers taken from Collins (2003))

- Subject-verb pairs: over 95% recall and precision
- Object-verb pairs: over 92% recall and precision
- Other arguments to verbs: ≈ 93% recall and precision
- Non-recursive NP boundaries: ≈ 93% recall and precision
- PP attachments: ≈ 82% recall and precision
- Coordination ambiguities: ≈ 61% recall and precision