Natural Language Processing

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

Session Content

- Grammars and Sentence Structure
- Parsing
- A Bottom-Up Chart Parser
- Probabilistic Context-Free Grammars
- Probabilistic CKY Parsing of PCFGs
- Ways to Learn PCFG Rule Probabilities
- Problems with PCFGs
- Parser Evaluation
- Code Demo

Ambiguity is Explosive

- "I saw the man with the telescope": 2 parses
- "I saw the man on the hill with the telescope.": 5 parses
- "I saw the man on the hill in Texas with the telescope": 14 parses
- "I saw the man on the hill in Texas with the telescope at noon.": 42 parses
- "I saw the man on the hill in Texas with the telescope at noon on Monday" 132 parses

Some funny examples

innovate achieve lead

– Policeman to little boy: “We are looking for a thief with a bicycle.” Little boy: “Wouldn’t you be better using your eyes.”
– Why is the teacher wearing sun-glasses. Because the class is so bright.

Grammars and Sentence Structure

- **Grammar**- formal specification of the structures allowable in the language, and
- **Parsing technique-** method of analyzing a sentence to determine its structure according to the grammar

What Makes a Good Grammar

- In small grammars, such as those that describe only a few types of sentences, one structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other.
- The analysis that retains its simplicity and generality as it is extended is more desirable
- Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs).

Context Free Grammar

N a set of non-terminal symbols (or variables)

Σ a set of terminal symbols (disjoint from N)

R a set of productions or rules of the form $A \rightarrow \beta$, where A is a non-terminal and β is a string of symbols from $(\Sigma \cup N)^*$

S, a designated non-terminal called the start symbol

Categories of Phrases

innovate achieve lead

Noun phrase (NP): Noun acts as the head word. They start with an article or noun.

Verb phrase (VP): Verb acts as the head word. They start with an verb

Adjective phrase (ADJP): Adjective as the head word. They start with an adjective

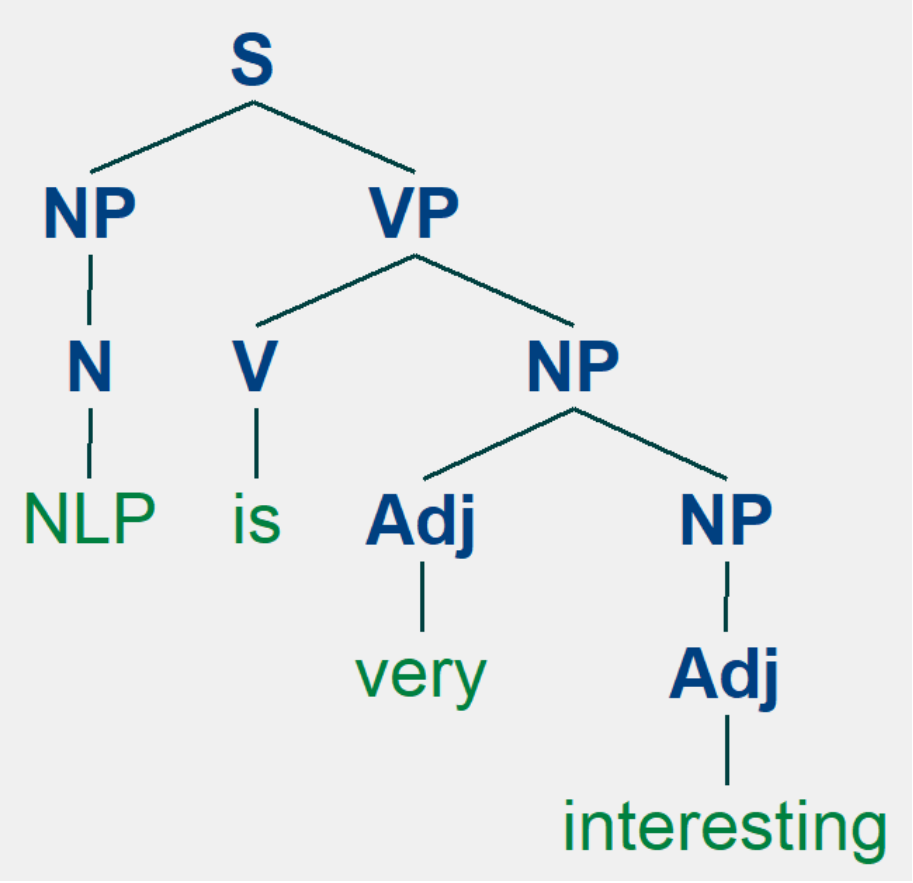
Adverb phrase (ADVP): Adverb acts as the head word. They usually start with an adverb

Prepositional phrase (PP): Preposition as the head word. They start with an preposition.

Simple grammar and Parsing Example

S -> NP VP
VP -> V NP
NP -> N
NP -> Adj NP
NP -> Adj

N -> NLP
V -> is
Adj -> very
Adj -> interesting



Applications of Parsing

innovate achieve lead

Sentiment Analysis: Compare "I like Frozen", "I do not like Frozen", and "I like frozen yogurt". The three sentences' words are very similar to each other, yet the first and the second contain inverse statements about the movie "Frozen", while the third is a statement about something else. Parsing here is crucial for understanding.

Relation Extraction: "Rome is the capital of Italy and the region of Lazio". While entity extraction can give us the entities here, we need parsing to see which entity is the capital of which other entities.

Question Answering: When answering "Who was the first man in space?" you need to parse the question and use parsed sentences to build the answer.

Applications of Parsing

Speech Recognition: Parsing scores the strings with either a pass/fail or a likelihood score, to give a powerful language model for speech recognition. Similarly, parsing could help in **spell checking, optical character recognition (OCR), text prediction, handwriting detection** etc.

Machine Translation: Parsing allows us to choose between several possible translations. Also, it makes translation of phrases and terms easier.

Grammar Checking: Parsing can help in checking the grammaticality of a document.

Parsing

Given a string of non-terminals and a CFG, determine if the string can be generated by the CFG.

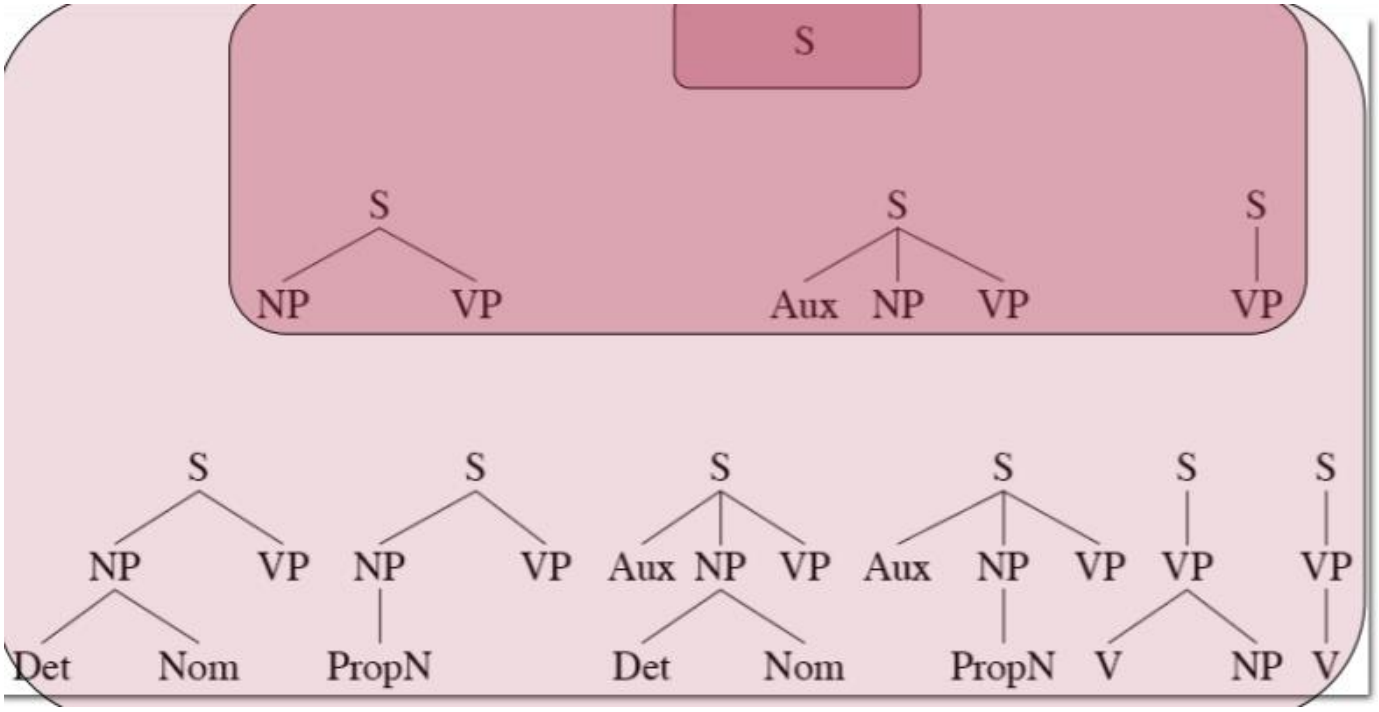
- Also return a parse tree for the string
- Also return all possible parse trees for the string

Must search space of derivations for one that derives the given string.

- Top-Down Parsing: Start searching space of derivations for the start symbol.
- Bottom-up Parsing: Start search space of reverse deivations from the terminal symbols in the string.

Top down Parsing

- **Parse tree is generated in the top to bottom fashion from root to leaves**
- **Search space from the start symbol sentence S on LHS**



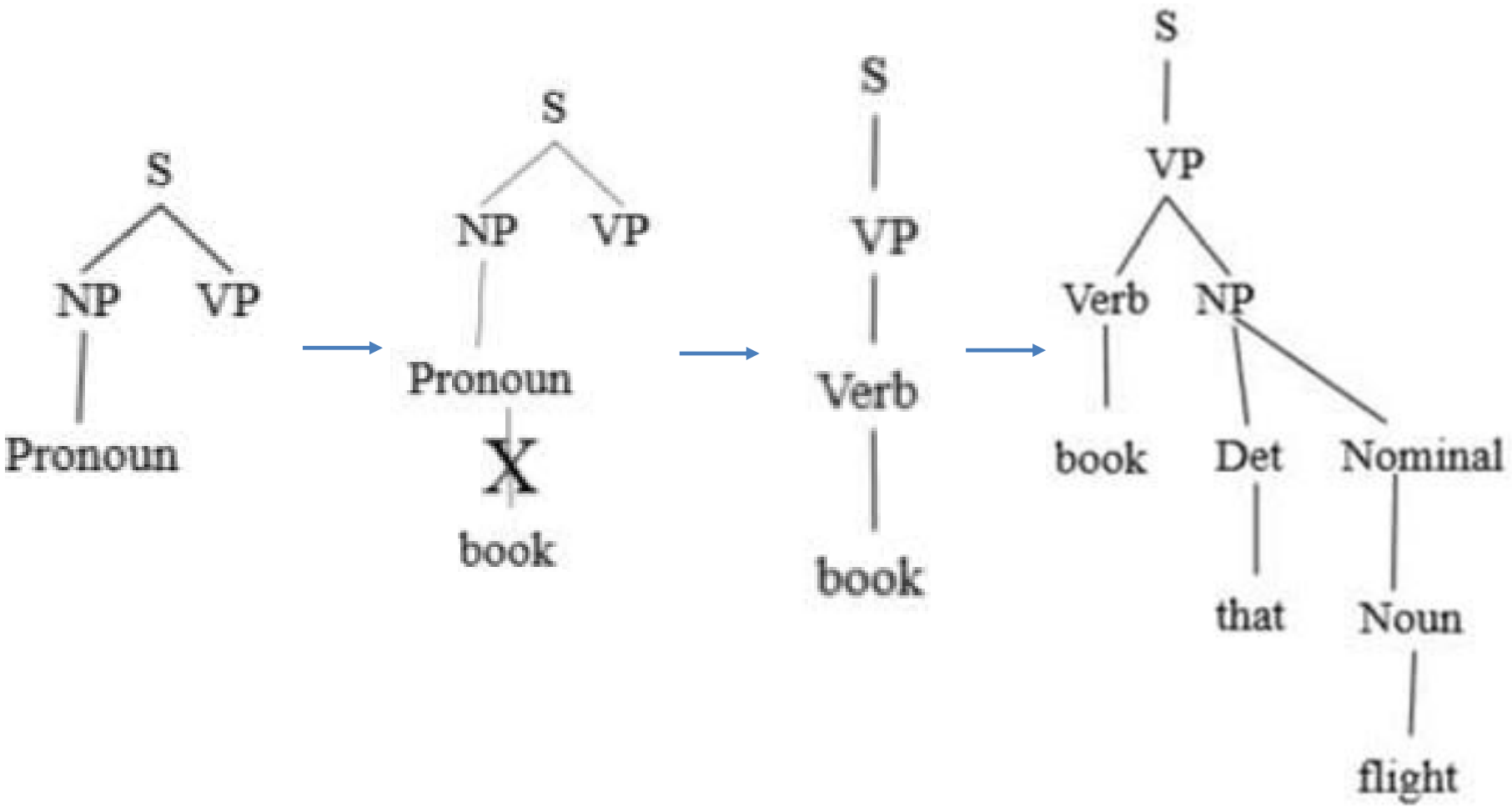
Top down parsing

1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N

4. VP -> V

5. VP -> V NP

Top down parsing



Bottom up parsing

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules.

You could build a bottom-up parser simply by formulating this matching process as a search process.

The state would simply consist of a symbol list, starting with the words in the sentence.

Successor states could be generated by exploring all possible ways to

- Rewrite a word by its possible lexical categories
- Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol

Bottom up parsing

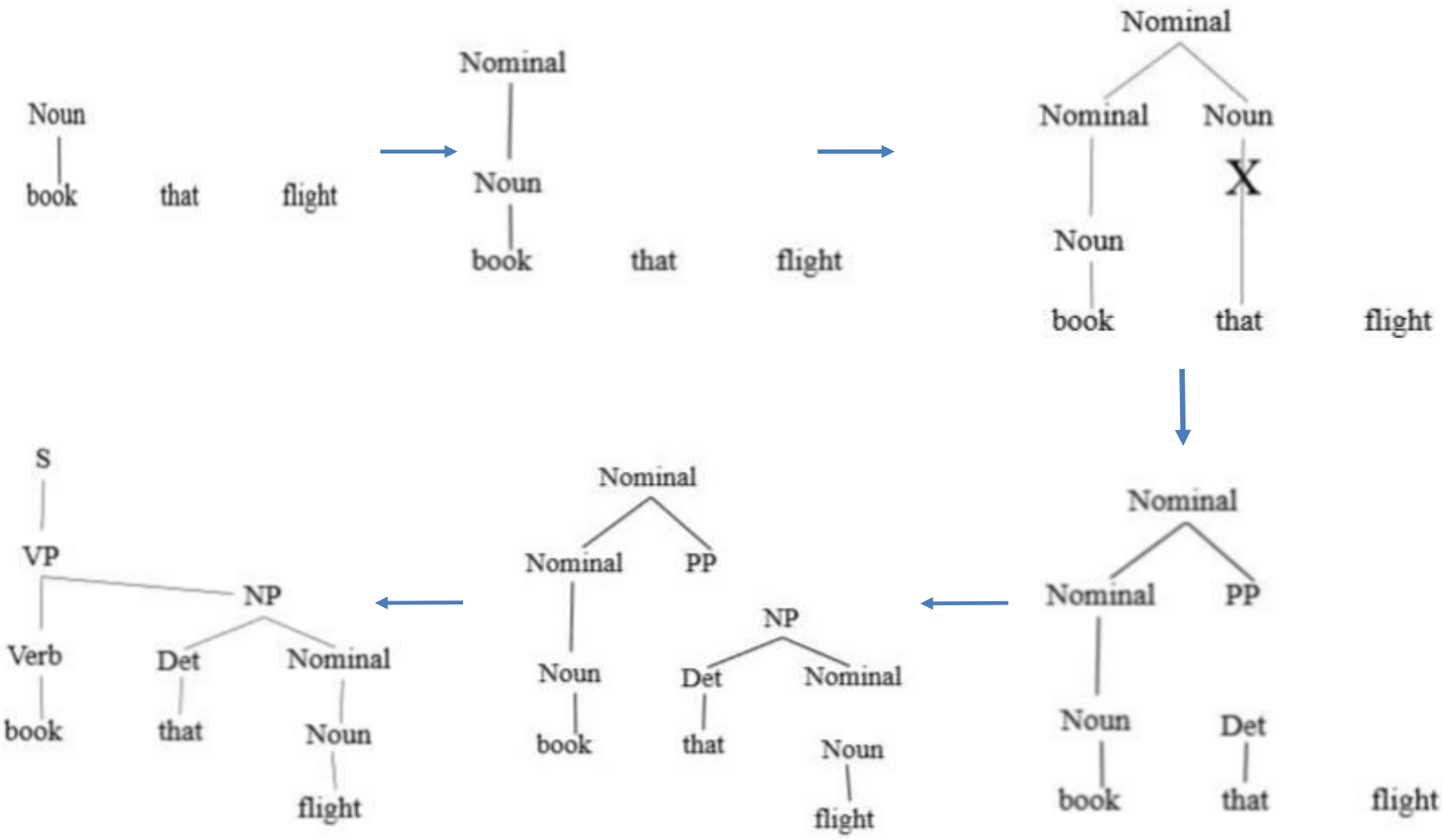


Chart parsing

- Parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily.
- To avoid this problem, a data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Chart Parsing

- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to "match"
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

Chart Parsing

- Assume you are parsing a sentence that starts with an ART.
- With this ART as the key, rules 2 and 3 are matched because they start with ART.
- To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART.
- You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record

  2'. NP -> ART o ADJ N

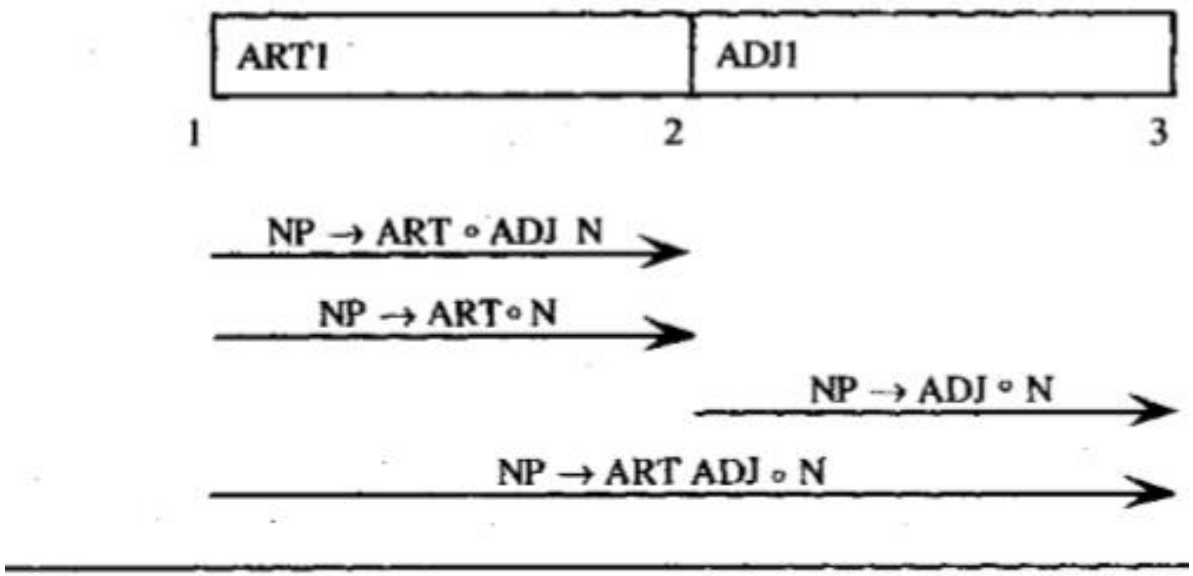
  3'. NP -> ART o N

- If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give

  2''. NP -> ART ADJ o N

Chart- active arcs

- The chart maintains the record of all the constituents derived from the sentence so far in the parse.
- Maintains the record of rules that have matched partially but are not complete. These are called the active arcs.



1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N

4. VP -> V

5. VP -> V NP

Figure 3.9 The chart after seeing an ADJ in position 2

Chart Parsing

Extending Active Arcs with a Key:

- Each **Active Arc** has the form: $[A \rightarrow X_1 ... \bullet C ... X_m](p_i, p_j)$
- A Key constituent has the form: $C(p_i, p_j)$
- When processing the Key $C(p_1, p_2)$, we search the active arc list for an arc $[A \rightarrow X_1 ... \bullet C ... X_m](p_0, p_1)$, and then create a new active arc $[A \rightarrow X_1 ... C \bullet ... X_m](p_0, p_2)$
- If the new active arc is a completed rule: $[A \rightarrow X_1 ... C \bullet](p_0, p_2)$, then we add $A(p_0, p_2)$ to the Agenda
- After "using" the key to extend all relevant arcs, it is entered into the Chart

Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time

1. Find all POS of word (terminal-level)
2. Initialize Agenda with all POS of the word
3. Pick a Key from the Agenda
4. Add all grammar rules that start with the Key as active arcs
5. Extend any existing active arcs with the Key
6. Add LHS constituents of newly completed rules to the Agenda
7. Add the Key to the Chart
8. If Agenda not empty - goto (3), else goto (1)

Chart parsing example

The input: "x = The large can can hold the water"

POS of Input Words:

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

$$
\begin{aligned}
(1)\ S &\rightarrow NP\ VP \\
(2)\ NP &\rightarrow ART\ ADJ\ N \\
(3)\ NP &\rightarrow ART\ N \\
(4)\ NP &\rightarrow ADJ\ N \\
(5)\ VP &\rightarrow AUX\ VP \\
(6)\ VP &\rightarrow V\ NP
\end{aligned}
$$

Chart parsing example

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

The input: "$x =$ **The** large can can hold the water"

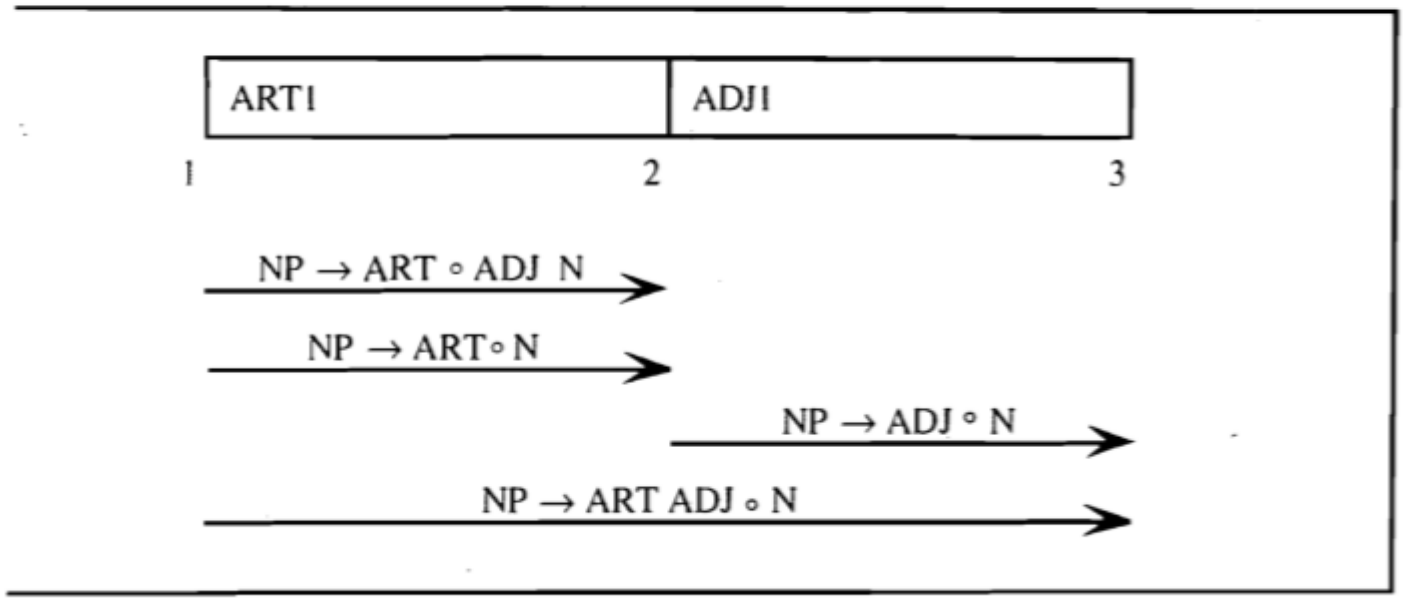


Figure 3.9 The chart after seeing an ADJ in position 2

Chart parsing example

The input: "x = **The large can** can hold the water"

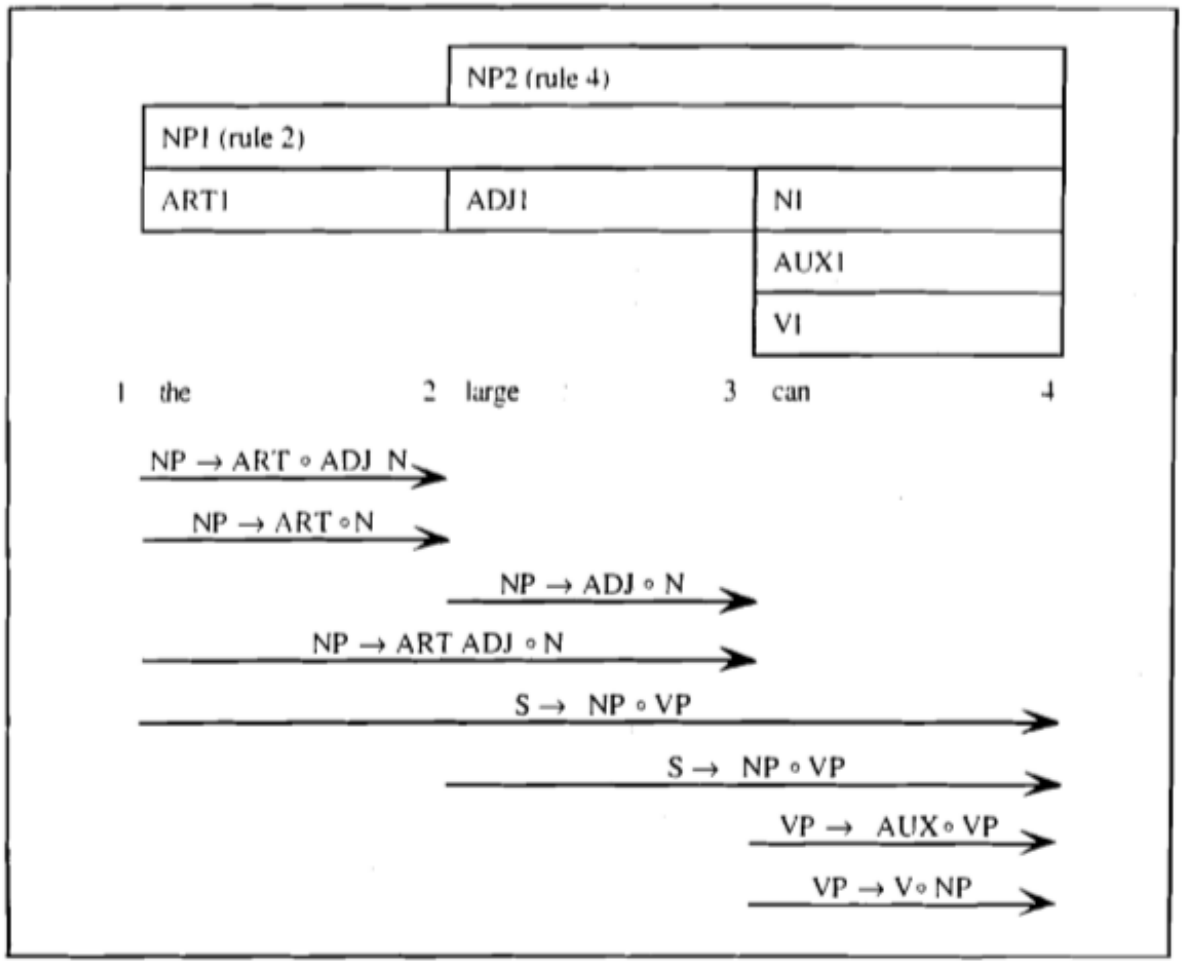


Figure 3.12 After parsing *the large can*

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

Chart parsing example

The input: "$x =$ **The large can can hold** the water"

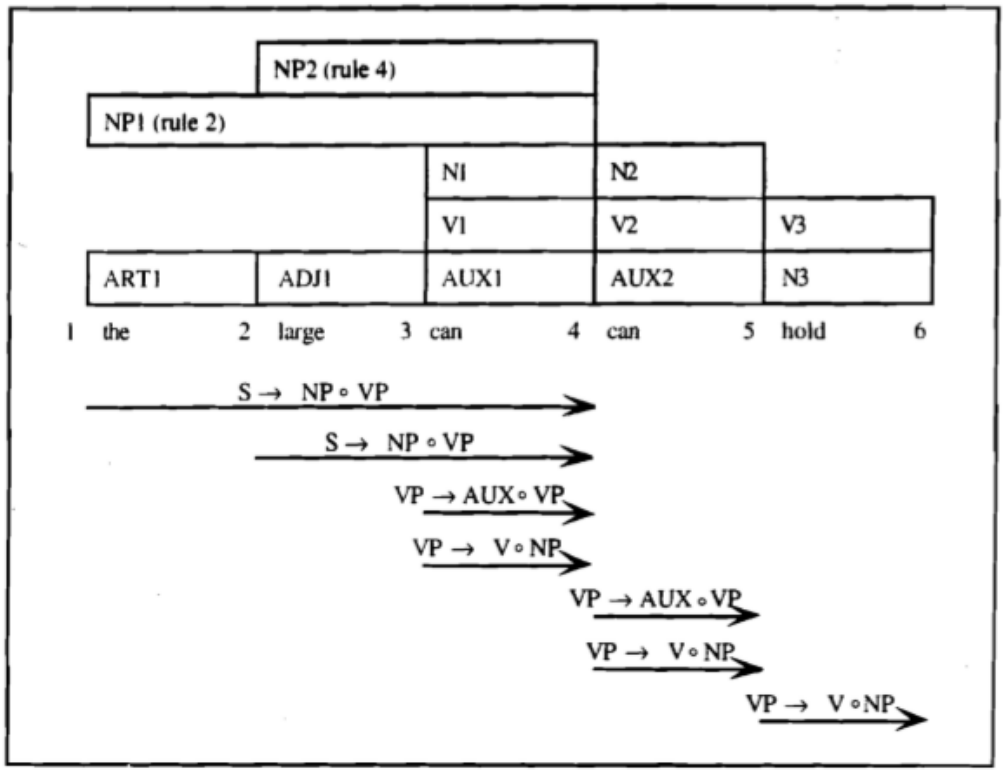


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

(1) $S \rightarrow NP\ VP$

(2) $NP \rightarrow ART\ ADJ\ N$

(3) $NP \rightarrow ART\ N$

(4) $NP \rightarrow ADJ\ N$

(5) $VP \rightarrow AUX\ VP$

(6) $VP \rightarrow V\ NP$

Chart parsing example

The input: "x = **The large can can hold the water**"

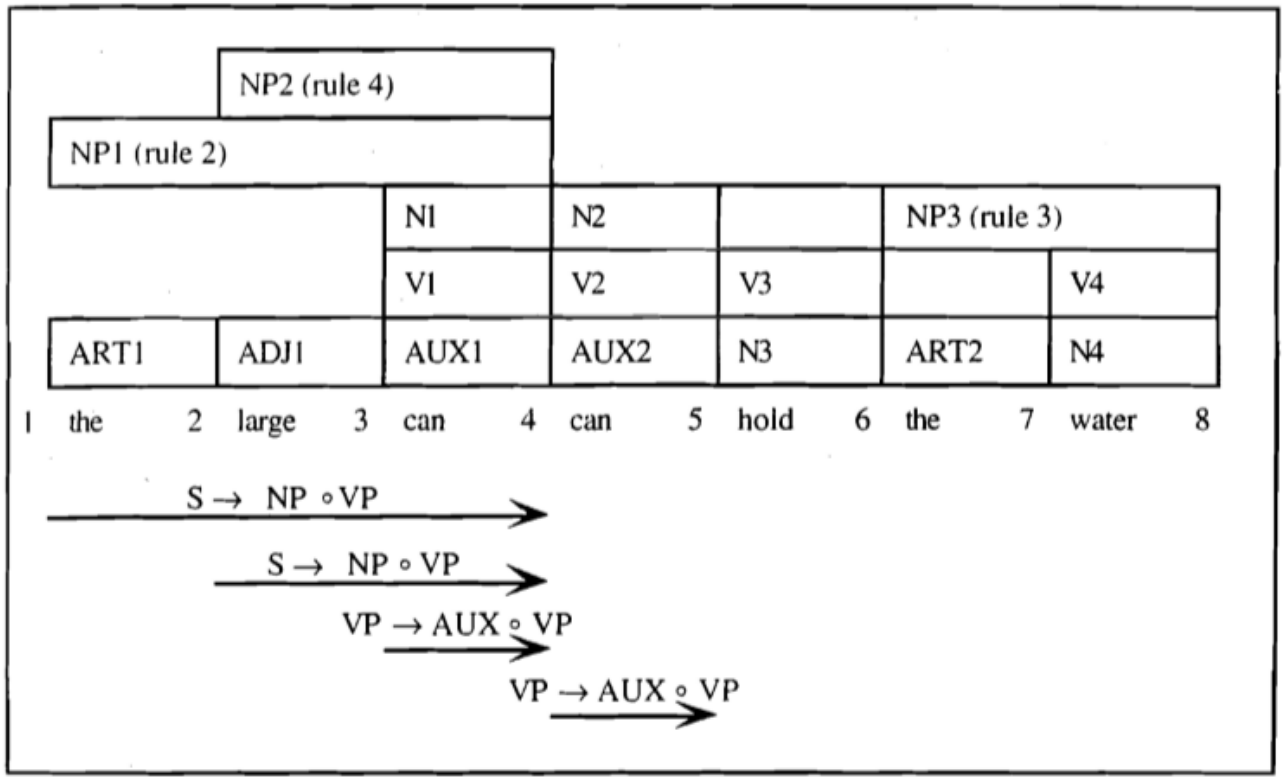


Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

Final Chart

The input: "$x =$ **The large can can hold the water**"

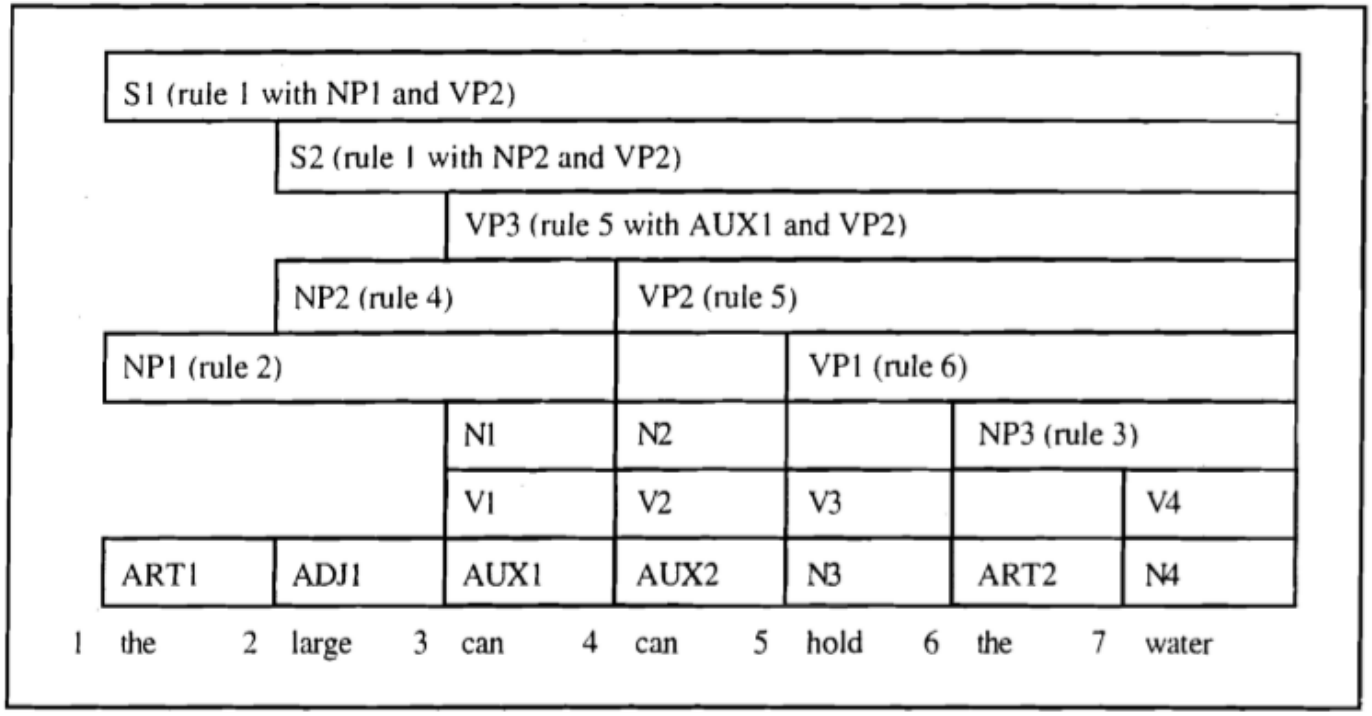


Figure 3.15 The final chart

(1) $S \rightarrow NP\ VP$
(2) $NP \rightarrow ART\ ADJ\ N$
(3) $NP \rightarrow ART\ N$
(4) $NP \rightarrow ADJ\ N$
(5) $VP \rightarrow AUX\ VP$
(6) $VP \rightarrow V\ NP$

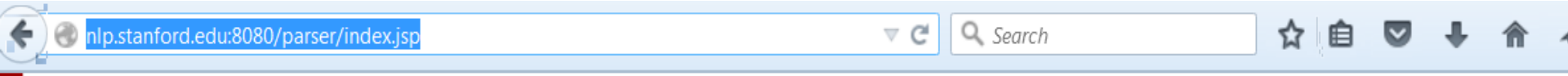
- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

Chart Parsing

- When a chart parser begins parsing a text, it creates a new (empty) chart, spanning the text.
- It then incrementally adds new edges to the chart.
- A set of "chart rules" specifies the conditions under which new edges should be added to the chart.
- Once the chart reaches a stage where none of the chart rules adds any new edges, parsing is complete.

Advantages of Chart Parsing

- No repeated computation of same sub problem
- Deals well with left-recursive grammars
- Deals well with ambiguity
- No backtracking necessary



Stanford Parser

Please enter a sentence to be parsed:

I love natural language processing class

Language: English Sample Sentence Parse

Your query

I love natural language processing class

Tagging

```
I/PRP  love/VBP  natural/JJ  language/NN  processing/NN  class/NN
```

Parse

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBP love)
      (NP (JJ natural) (NN language) (NN processing) (NN class)))))
```

Formal Definition of a PCFG

- A PCFG consists of:
  - A set of terminals, $\{w^k\}$, k= 1,…,V
  - A set of nonterminals, N^i, i= 1,…, n
  - A designated start symbol N^1
  - A set of rules, $\{N^i \rightarrow \xi^j\}$, (where ξ^j is a sequence of terminals and nonterminals)
  - A corresponding set of probabilities on rules such that: $\forall i \quad \Sigma_j P(N^i \rightarrow \xi^j) = 1$

Probability of a Derivation Tree and a String

- The probability of a derivation (i.e. parse) tree:

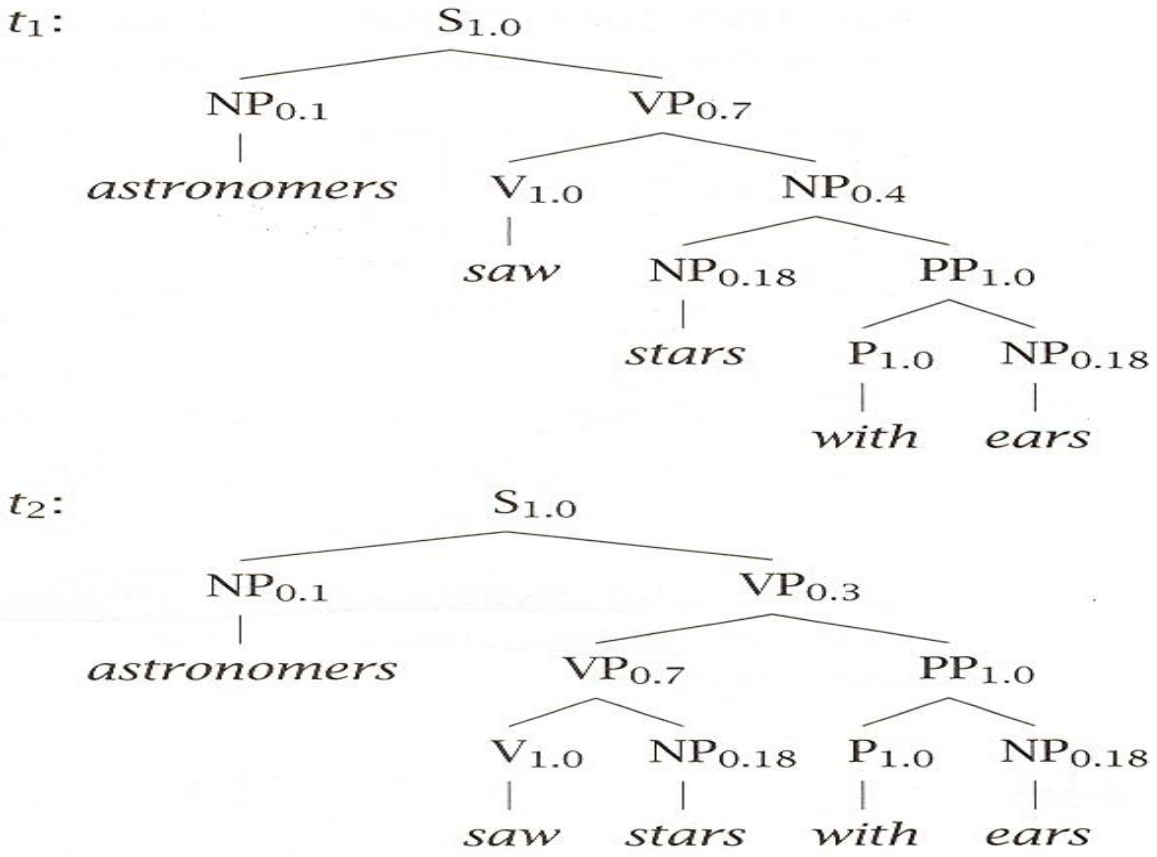
  $$P(T) = \Pi_{i=1..k}\, p(r(i))$$

  where r(1), …, r(k) are the rules of the CFG used to generate the sentence $\mathbf{w_{1m}}$ of which T is a parse.
- The probability of a sentence (according to grammar G) is given by:

  $$P(w_{1m}) = \Sigma_t\, P(w_{1m}, t) = \Sigma_{\{t:\, yield(t)=w_{1m}\}}\, P(t)$$

  where t is a parse tree of the sentence. Need dynamic programming to make this efficient!

Example: Probability of a Derivation Tree



S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

Some Features of PCFGs

- A PCFG gives some idea of the plausibility of different parses; however, the probabilities are based on structural factors and not lexical ones.
- PCFGs are good for grammar induction.
- PCFGs are robust.
- PCFGs give a probabilistic language model for English.
- The predictive power of a PCFG tends to be greater than for an HMM.
- PCFGs are not good models alone but they can be combined with a trigram model.

Properties of PCFGs

- Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- Say we have a sentence s, set of derivations for that sentence is $T(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $T(s)$. i.e., *we now have a ranking in order of probability*.
- The most likely parse tree for a sentence s is

$$\arg\max_{t \in T(s)} p(t)$$

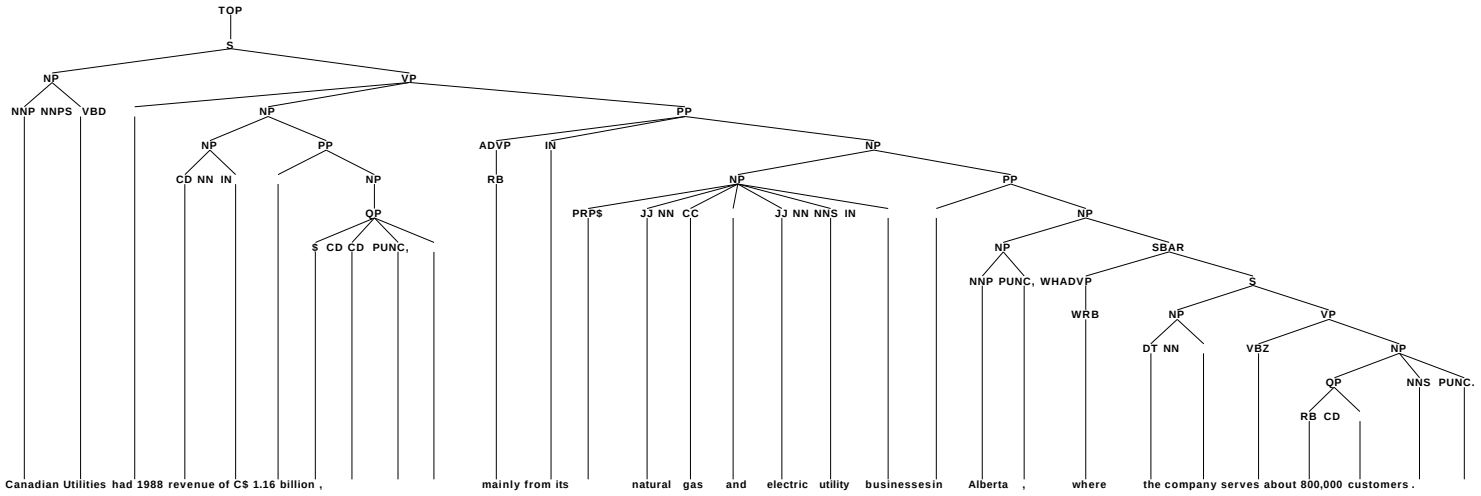
PCFG based Grammar

- PCFGs augments CFGs by including a probability for each rule in the grammar.
- The probability for a parse tree is the product of probabilities for the rules in the tree
- To build a PCFG-parsed parser:
  1. Learn a PCFG from a treebank
  2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

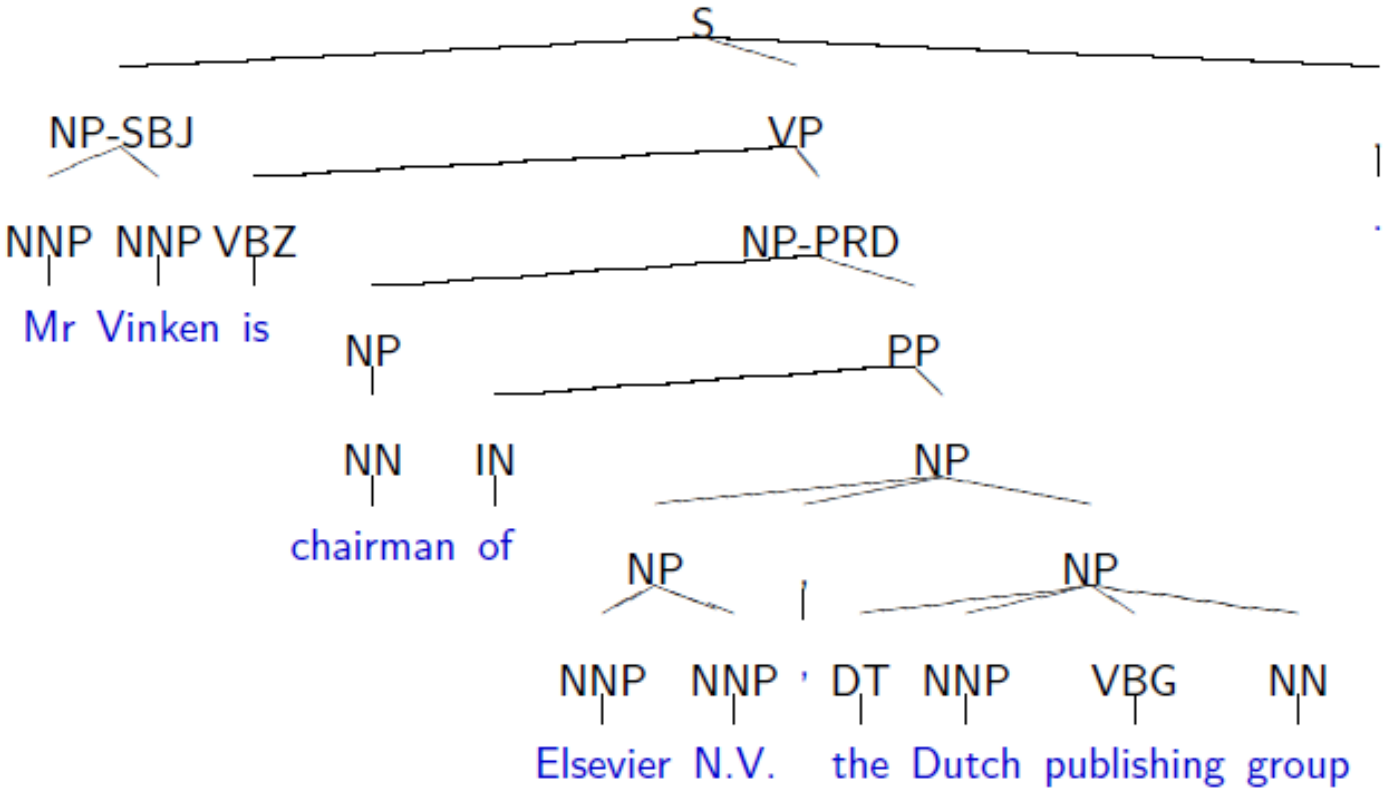
Data for Parsing Experiments: Treebanks

- Penn WSJ Treebank = 50,000 sentences with associated trees
- Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



Example tree



Characteristics of PCFGs

- In a PCFG, the probability $P(A \rightarrow \beta)$ expresses the likelihood that the non-terminal A will expand as β.
  - e.g. the likelihood that S → NP VP
    - (as opposed to S→VP, or S → NP VP PP, or…)

- can be interpreted as a conditional probability:
  - probability of the expansion, given the LHS non-terminal
  - $P(A \rightarrow \beta) = P(A \rightarrow \beta | A)$

- Therefore, for any non-terminal A, probabilities of every rule of the form A → β must sum to 1
  - in this case, we say the PCFG is consistent

Word/Tag Counts

	N	V	ART	P	TOTAL
flies	21	23	0	0	44
fruit	49	5	1	0	55
like	10	30	0	21	61
a	1	0	201	0	202
the	1	0	300	2	303
flower	53	15	0	0	68
flowers	42	16	0	0	58
birds	64	1	0	0	65
others	592	210	56	284	1142
TOTAL	833	300	558	307	1998

Lexical Probability Estimates

P(the|ART)=300 /558 = 0.54

P(the|ART)	.54	P(a|ART)	.36
P(flies|N)	.025	P(a|N)	.001
P(flies|V)	.076	P(flower|N)	.063
P(like|V)	.1	P(flower|V)	.05
P(like|P)	.068	P(birds|N)	.076
P(like|N)	.012		

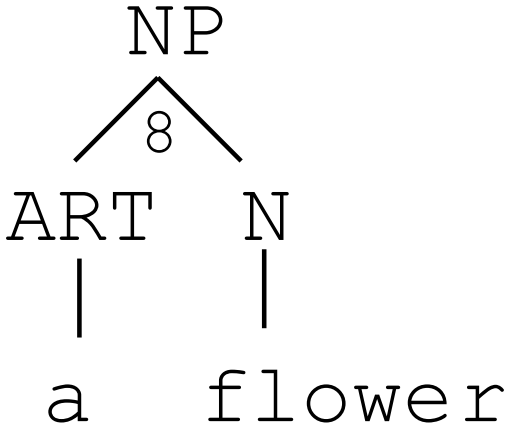
The PCFG

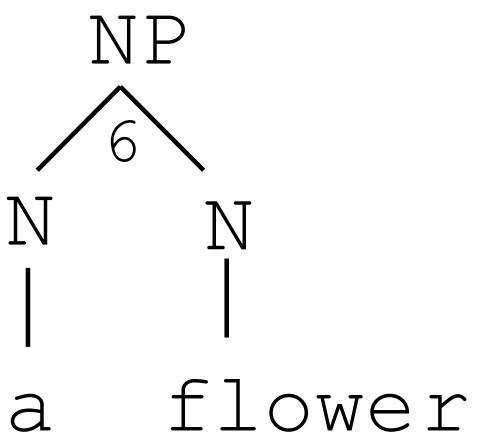
- Below is a probabilistic CFG (PCFG) with probabilities derived from analyzing a parsed version of Allen's corpus.

Rule	Count for LHS	Count for Rule	PROB
1. S → NP VP	300	300	1
2. VP → V	300	116	.386
3. VP → V NP	300	118	.393
4. VP → V NP PP	300	66	.22
5. NP → NP PP	1032	241	.23
6. NP → N N	1032	92	.09
7. NP → N	1032	141	.14
8. NP → ART N	1032	558	.54
9. PP → P NP	307	307	1

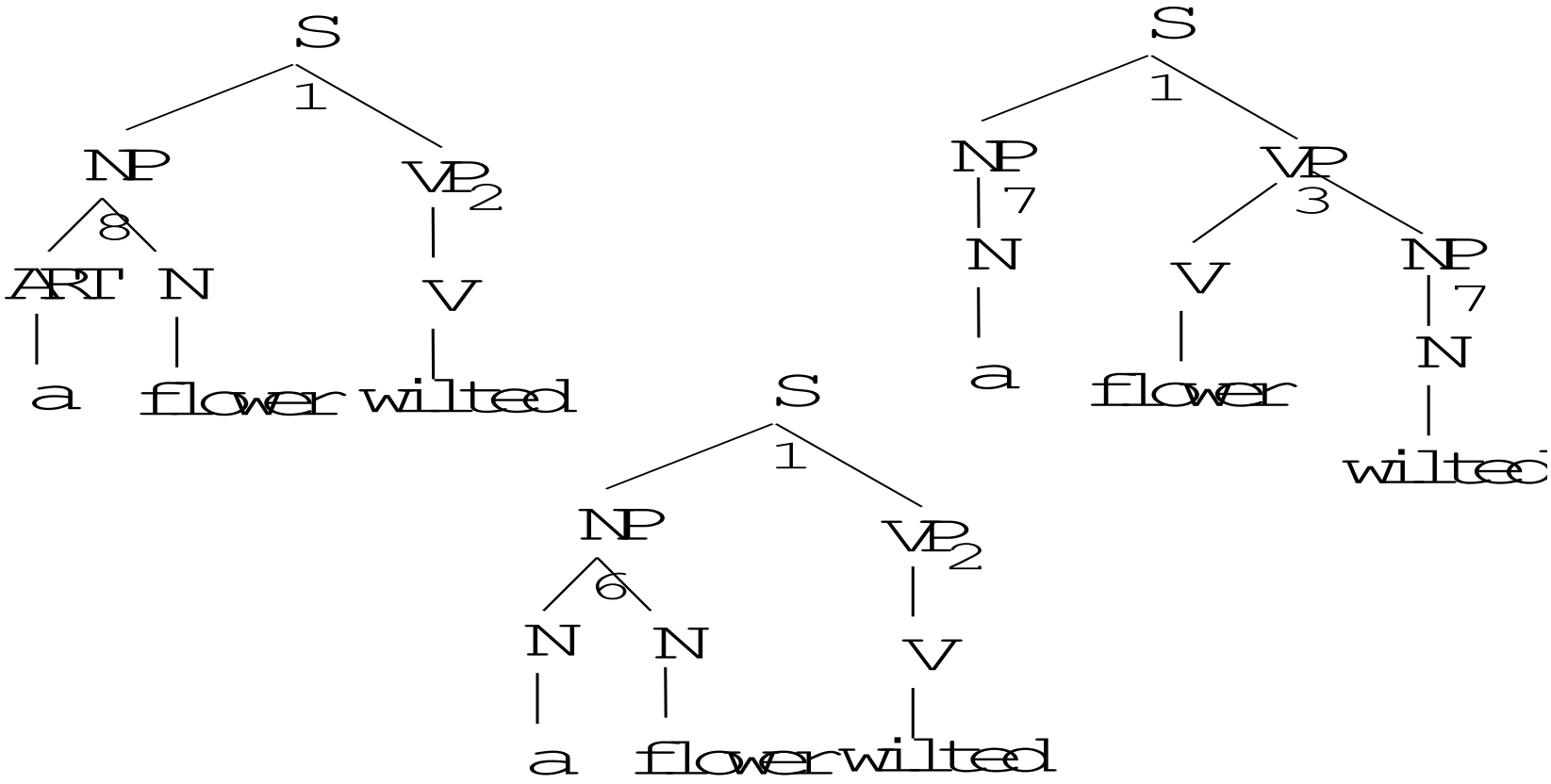
Parsing with a PCFG

- Using the lexical probabilities, we can derive probabilities that the constituent NP generates a sequence like *a flower*. Two rules could generate the string of words:





Three Possible Trees for an S



Parsing with a PCFG

- The probability of a sentence generating *A flower wilted*:

$$P(\textit{a flower wilted} \mid S) = P(R_1 \mid S) \times P(\textit{a flower} \mid NP) \times P(\textit{wilted} \mid VP) + P(R_1 \mid S) \times P(\textit{a} \mid NP) \times P(\textit{flower wilted} \mid VP)$$

- Using this approach, the probability that a given sentence will be generated by the grammar can be efficiently computed.
- It only requires some way of recording the value of each constituent between each two possible positions. The requirement can be filled by a packed chart structure.

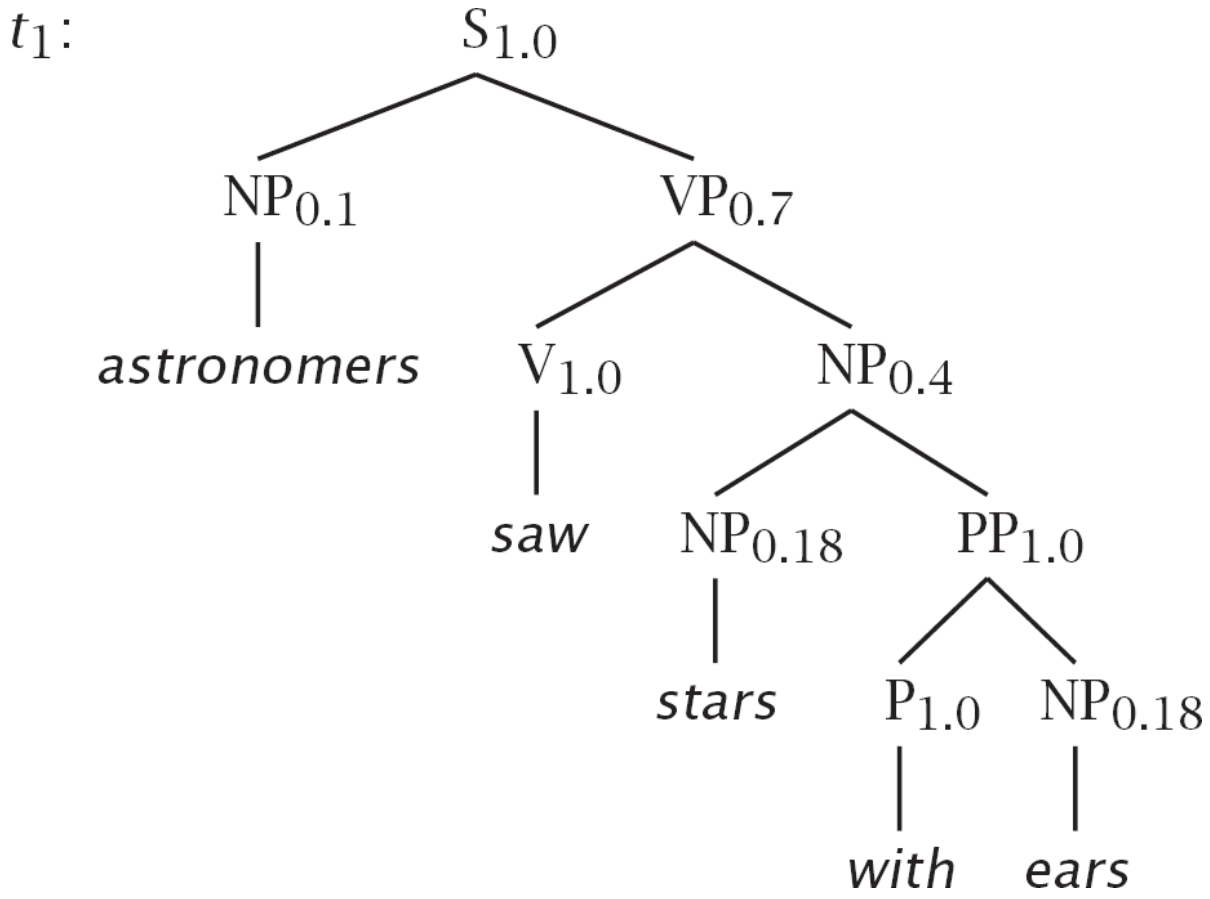
Example

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

- Terminals with, saw, astronomers,ears, stars, telescopes
- Nonterminals S, PP, P, NP, VP, V
- Start symbol S

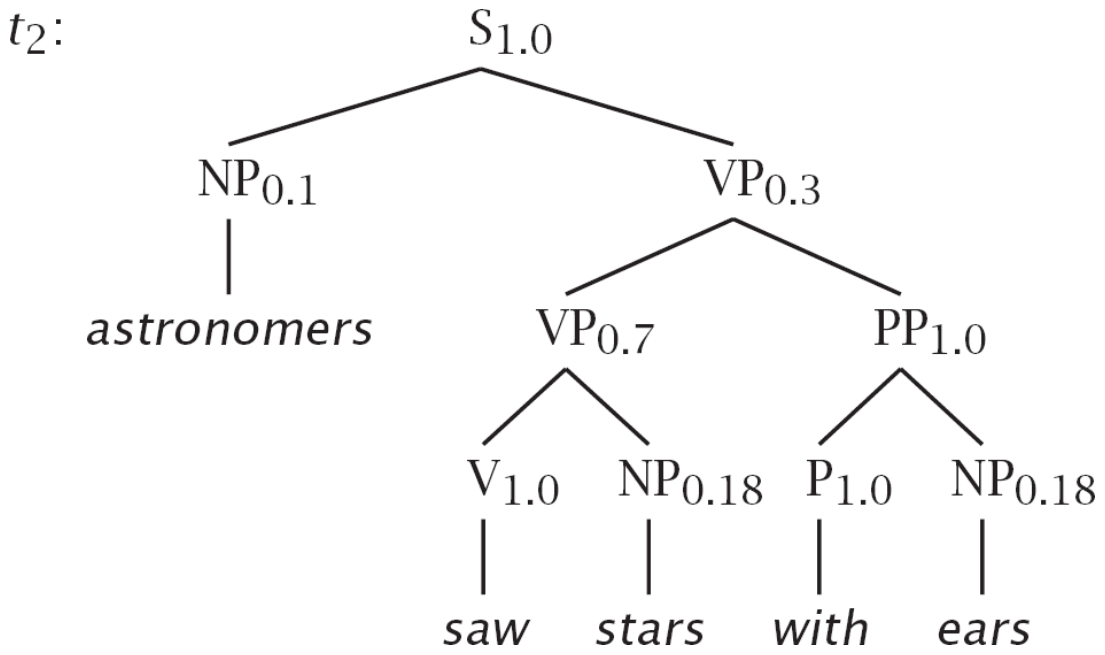
Slide based on "Foundations of Statistical Natural Language Processing" by Christopher Manning and Hinrich Schütze

astronomers saw stars with ears



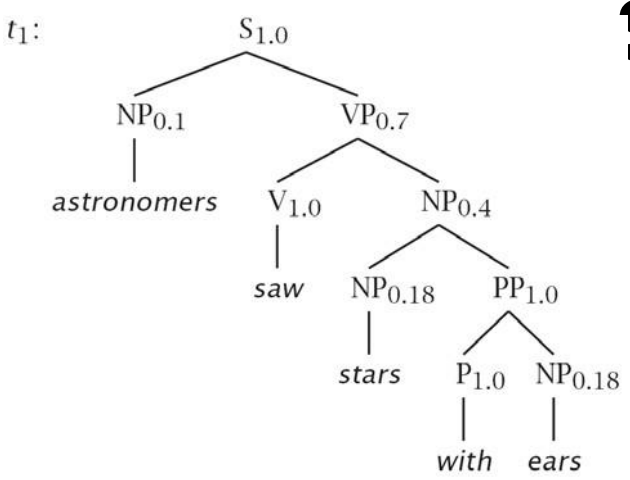
Slide based on “Foundations of Statistical Natural Language Processing” by Christopher Manning and Hinrich Schütze

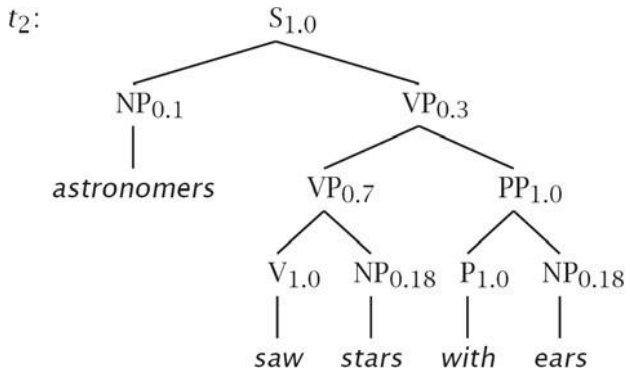
astronomers saw stars with ears



Slide based on “Foundations of Statistical Natural Language Processing” by Christopher Manning and Hinrich Schütze

Probabilities





$$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0009072$$

$$P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0006804$$

$$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$$

Slide based on “Foundations of Statistical Natural Language Processing” by Christopher Manning and Hinrich Schütze

Uses of probabilities in parsing

- Disambiguation: given *n* legal parses of a string, which is the most likely?
  - e.g. PP-attachment ambiguity can be resolved this way
- Speed: we've defined parsing as a search problem
  - search through space of possible applicable derivations
  - search space can be pruned by focusing on the most likely sub-parses of a parse
- Parser can be used as a model to determine the probability of a sentence, given a parse
  - typical use in speech recognition, where input utterance can be "heard" as several possible sentences

Using PCFG probabilities

- PCFG assigns a probability to every parse-tree t of a string W
  - e.g. every possible parse (derivation) of a sentence recognised by the grammar

- Notation:
  - G = a PCFG
  - s = a sentence
  - t = a particular tree under our grammar
    - t consists of several nodes *n*
    - each node is generated by applying some rule *r*

Probability of a tree vs. a sentence

- We work out the probability of a parse tree t by multiplying the probability of every rule (node) that gives rise to t (i.e. the derivation of t).

- Note that:
  - A tree can have multiple derivations
    - (different sequences of rule applications could give rise to the same tree)
  - But the probability of the tree remains the same
    - (it's the same probabilities being multiplied)
  - We usually speak as if a tree has only one derivation, called the **canonical derivation**

Picking the best parse in a PCFG

- A sentence will usually have several parses
  - we usually want them ranked, or only want the *n* best parses
  - we need to focus on P(t|s,G)
    - probability of a parse, given our sentence and our grammar
  - definition of the best parse for s:
    - The tree for which P(t|s,G) is highest

Probability of a sentence

- Given a probabilistic context-free grammar G, we can the probability of a sentence (as opposed to a tree).

- Observe that:
  - As far as our grammar is concerned, a sentence is only a sentence if it can be recognised by the grammar (it is "legal")
  - There can be multiple parse trees for a sentence.
    - Many trees whose **yield** is the sentence
  - The probability of the sentence is the sum of all the probabilities of the various trees that yield the sentence.

Using CKY to parse with a PCFG

- The basic CKY algorithm remains unchanged.
- However, rather than only keeping partial solutions in our table cells (i.e. The rules that match some input), we also keep their probabilities.

CKY parsing

Classic, bottom-up dynamic programming algorithm (Cocke-Kasami-Younger).

Requires input grammar based on Chomsky Normal Form

- A CNF grammar is a Context-Free Grammar in which:
  - Every rule LHS is a non-terminal
  - Every rule RHS consists of either a single terminal or two non-terminals.
  - Examples:
    - A → B C
    - NP → N PP
    - A → a
    - Noun → man
  - But not:
    - NP → the N
    - S →VP

Chomsky Normal Form

- Any CFG can be re-written in CNF, without any loss of expressiveness.
  - That is, for any CFG, there is a corresponding CNF grammar which accepts exactly the same set of strings as the original CFG.
  - Normal forms give us more structure to work with, resulting in easier parsing algorithms.
  - CNF provides an upper bound for parsing **complexity**

Converting a CFG to CNF

- To convert a CFG to CNF, we need to deal with three issues:
  1. Rules that mix terminals and non-terminals on the RHS
     - E.g. NP → *the* Nominal
  2. Rules with a single non-terminal on the RHS (called unit productions)
     - E.g. NP → Nominal
  3. Rules which have more than two items on the RHS
     - E.g. NP → Det Noun PP

*Nominal definition is a noun, noun phrase, or any word or word group that functions as a noun. The term comes from the Latin, meaning "name."

Converting a CFG to CNF

1. Rules that mix terminals and non-terminals on the RHS
   - E.g. NP → *the* Nominal
   - Solution:
     - Introduce a dummy non-terminal to cover the original terminal
       - E.g. Det → the
     - Re-write the original rule:
       - NP → Det Nominal
       - Det → the

Converting a CFG to CNF

2. Rules with a single non-terminal on the RHS (called unit productions)
   - E.g. NP → Nominal
   - Solution:
     - Find all rules that have the form Nominal → ...
       - Nominal → Noun PP
       - Nominal → Det Noun
     - Re-write the above rule several times to eliminate the intermediate non-terminal:
       - NP → Noun PP
       - NP → Det Noun
   - Note that this makes our grammar “flatter”

Converting a CFG to CNF

3. Rules which have more than two items on the RHS
   - E.g. NP → Det Noun PP

- Solution:
  - Introduce new non-terminals to spread the sequence on the RHS over more than 1 rule.
    - Nominal → Noun PP
    - NP → Det Nominal

Probabilistic CKY: example PCFG

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

Probabilistic CYK: initialisation

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	2	3	4	5
0					
1					
2					
3					
4					
5					

Probabilistic CYK: lexical step

- *The flight includes a meal.*

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

	1	2	3	4	5
0	Det (.4)				
1					
2					
3					
4					
5					

Probabilistic CYK: lexical step

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	2	3	4	5
0	Det (.4)				
1		N .02			
2					
3					
4					
5					

Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .3			
1		N .02			
2					
3					
4					
5					

Note: probability of NP in [0,2]
P(Det → the) * P(N → meal) * P(NP → Det N)

Probabilistic CYK: lexical step

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .3			
1		N .02			
2			V .05		
3					
4					
5					

Probabilistic CYK: lexical step

- *The flight includes a meal.*

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .3			
1		N .02			
2			V .05		
3				Det .4	
4					
5					

Probabilistic CYK: syntactic step

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .3			
1		N .02			
2			V .05		
3				Det .4	
4					N .01

Probabilistic CYK: syntactic step

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	**2**	**3**	**4**	**5**
0	Det (.4)	NP .3			
1		N .02			
2			V .05		
3				Det .4	NP .3
4					N .01

Probabilistic CYK: syntactic step

- *The flight includes a meal.*
  - S → NP VP [.80]
  - NP → Det N [.30]
  - VP → V NP [.20]
  - V → includes [.05]
  - Det → the [.4]
  - Det → a [.4]
  - N → meal [.01]
  - N → flight [.02]

	1	2	3	4	5
0	Det (.4)	NP .3			
1		N .02			
2			V .05		VP 0.2
3				Det .4	NP .3
4					N .01

Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- S → NP VP [.80]
- NP → Det N [.30]
- VP → V NP [.20]
- V → includes [.05]
- Det → the [.4]
- Det → a [.4]
- N → meal [.01]
- N → flight [.02]

	1	2	3	4	5
0	Det .4	NP .3			S (.8*.3*.4*.02 *.2*.05*.3*.4*.01)
1		N .02			
2			V .05		VP .2
3				Det .4	NP .3
4					N .01

Probabilistic CYK: summary

- Cells in chart hold probabilities
- Bottom-up procedure computes probability of a parse incrementally.
- To obtain parse trees, we traverse the table "backwards" as before.
  - Cells need to be augmented with backpointers.

Problems with PCFGs

- No Context
  - (immediate prior context, speaker, …)
- No Lexicalization
  - "VP NP NP" more likely if verb is "hand" or "tell"
  - fail to capture lexical dependencies (n-grams do)
- No StructuralContext
  - How NP expands depends on position

Slide based on "Foundations of Statistical Natural Language Processing" by Christopher Manning and Hinrich Schütze

Evaluating Parsers

- We need a measure to evaluate parser performance against gold standard
  - ratio of fully correct sentences parses too coarse
  - → ratio of correct constituents

- Does *correct* mean *precision*?

$$\text{precision} = \frac{\text{count(matching constituents)}}{\text{count(predicted constituents)}}$$

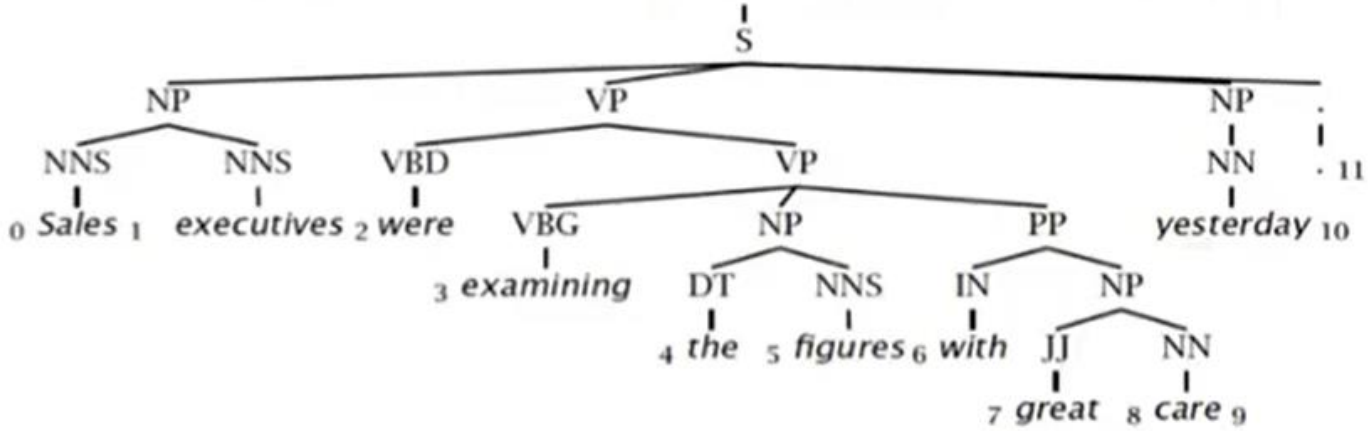
- Does *correct* mean *recall*?

$$\text{Recall} = \frac{\text{count(matching constituents)}}{\text{count(gold standard constituents)}}$$

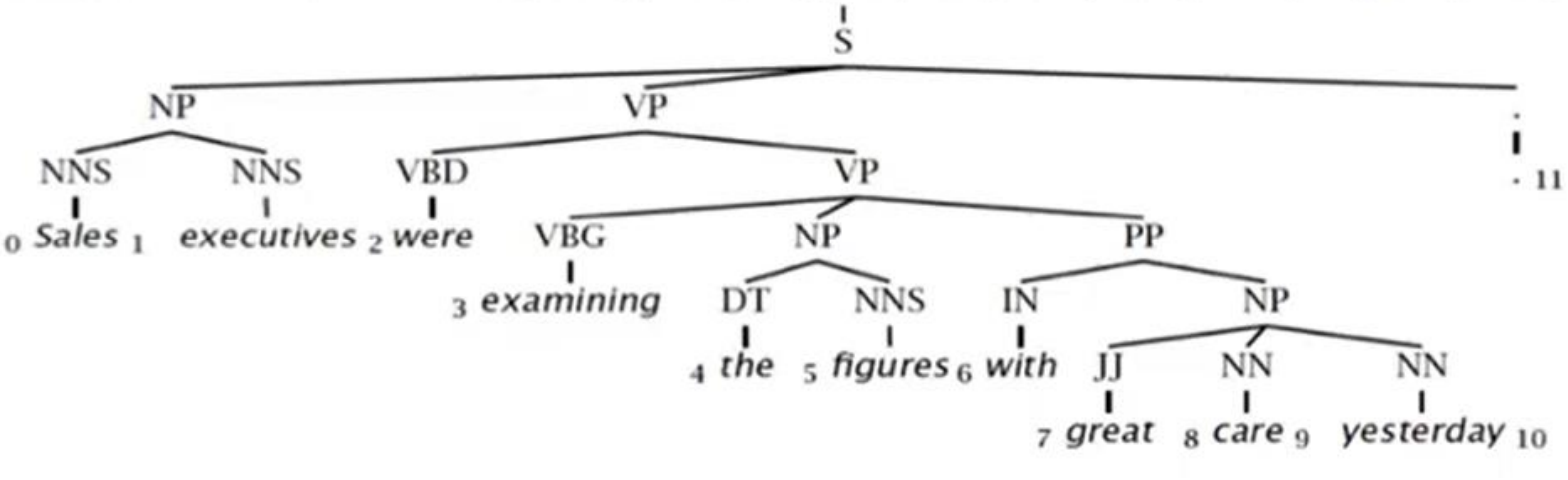
Credits: Philipp Koehn

Evaluating Parsers

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 Parsers

- **Gold standard brackets: 8**

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

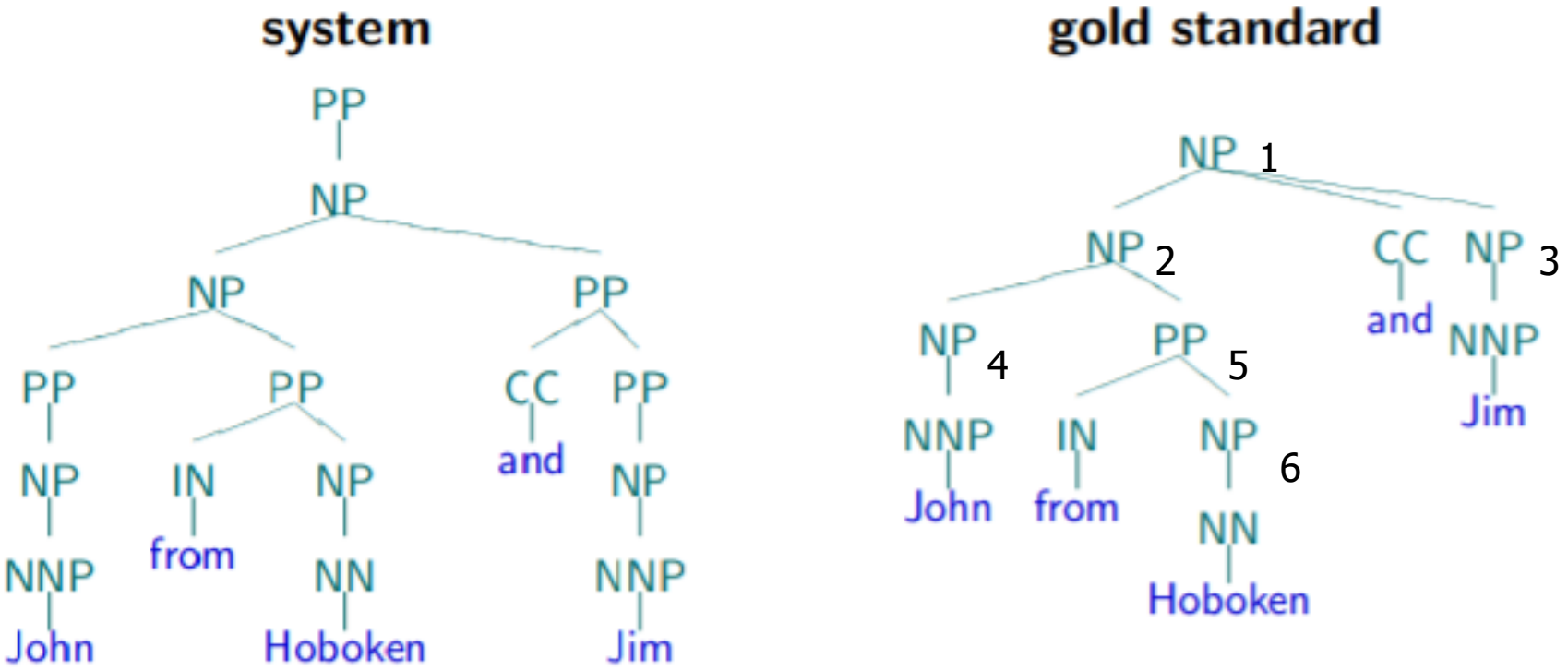
Candidate brackets: 7

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%

Evaluating Parsers

Low Precision, High Recall



all gold standard constituents are predicted (recall 6/6)
... but we are predicting many more (precision 6/10)

Credits: Philipp Koehn

Evaluating Parsers

- F-measure: balance of precision and recall

$$F_1 = \frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2}$$

- F-measure is used in many other NLP tasks and may be adjusted to give more emphasis to either precision or recall

Credits: Philipp Koehn

Extra Reading

- https://www.youtube.com/watch?v=Z6GsoBA-09k&list=PLQiyVNMpDLKnZYBTUOlSI9mi9wAErFtFm&index=62
- https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html
- http://www.nltk.org/howto/grammar.html
- https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_parsing.htm
- https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html
- http://courses.washington.edu/ling571/ling571_WIN2017/slides/ling571_class6_pcfg_impr_flat.pdf
- http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/lexpcfgs.pdf

References

- http://www.ai.mit.edu/courses/6.863/lecture7-03.pdf
- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin
- Natural language understanding by James Allen
- https://nlp.stanford.edu/software/lex-parser.shtml
- https://www.nltk.org/api/nltk.parse.html