BITS Pilani
Pilani Campus

Natural Language Processing
DSECL ZG565

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

innovate achieve lead

BITS Pilani
Pilani Campus

Session 10 -Statistical Constituency Parsing
Date – 17th Feb 2024

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin and many others who made their course materials freely available online.

Session Content

(Ref: Chapter 14 Jurafsky and Martin)

- CKY Parsing
- Probabilistic Context-Free Grammars
- PCFG for disambiguation
- Probabilistic CKY Parsing of PCFGs
- Ways to Learn PCFG Rule Probabilities
- Probabilistic Lexicalized CFGs
- Evaluating Parsers
- Problems with PCFGs

Some funny examples

– 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.

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

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

CNF Grammar

- If we parse a sentence with a CNF grammar, we know that:
  – Every phrase-level non-terminal (above the part of speech level) will have exactly 2 daughters.
    - NP → Det N
  – Every part-of-speech level non-terminal will have exactly 1 daughter, and that daughter is a terminal:
    - N → lady

Recognising strings with CKY

Example input: The flight includes a meal.

The CKY algorithm proceeds by:

1. Splitting the input into words and indexing each position.

   (0) the (1) flight (2) includes (3) a (4) meal (5)

2. Setting up a table. For a sentence of length n, we need (n+1) rows and (n+1) columns.

3. Traversing the input sentence left-to-right

4. Use the table to store constituents and their span.

CKY example

- S → NP VP
- NP → Det N
- VP → V NP
- V → includes
- Det → the
- Det → a
- N → meal
- N → flight

CKY example

Rule: Det → *the*

[0,1] for "the"

	1	2	3	4	5
0	**Det**				S
1					
2					
3					
4					
	the	flight	includes	a	meal

CKY example

Rule1: Det → *the*
Rule 2: N → flight

[0,1] for "the"   [1,2] for "flight"

	1	2	3	4	5
0	**Det**				S
1		**N**			
2					
3					
4					
	the	flight	includes	a	meal

CKY example

[0,2] for "the flight"

[0,1] for "the"

[1,2] for "flight"

Rule1: Det → *the*
Rule 2: N → flight
Rule 3: NP → Det N

	1	2	3	4	5
0	**Det**	**NP**			S
1		**N**			
2					
3					
4					
	the	flight	includes	a	meal

CKY: lexical step (j = 1)

- *The flight includes a meal.*

Lexical lookup
- Matches Det → the

	1	2	3	4	5
0	Det				
1					
2					
3					
4					
5					

CKY: lexical step (j = 2)

- *The flight includes a meal.*

Lexical lookup
- Matches N → flight

	1	2	3	4	5
0	Det				
1		N			
2					
3					
4					
5					

CKY: syntactic step (j = 2)

- *The flight includes a meal.*

Syntactic lookup:
• look backwards and see if there is any rule that will cover what we've done so far.

	1	2	3	4	5
0	Det	NP			
1		N			
2					
3					
4					
5					

CKY: lexical step (j = 3)

- *The flight includes a meal.*

Lexical lookup
- Matches V → includes

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					
5					

CKY: lexical step (j = 3)

- *The flight includes a meal.*

Syntactic lookup
• There are no rules in our grammar that will cover Det, NP, V

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					
5					

CKY: lexical step (j = 4)

- *The flight includes a meal.*

Lexical lookup
• Matches Det → a

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					
5					

CKY: lexical step (j = 5)

- *The flight includes a meal.*

Lexical lookup
- Matches N → meal

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal.*

Syntactic lookup
• We find that we have
NP → Det N

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	NP
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal.*

Syntactic lookup
• We find that we have
VP → V NP

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		VP
3				Det	NP
4					N

CKY: syntactic step (j = 5)

- *The flight includes a meal*.

Syntactic lookup

• We find that we have

S → NP VP

	1	2	3	4	5
0	Det	NP			S
1		N			
2			V		VP
3				Det	NP
4					N

From recognition to parsing

- The procedure so far will recognise a string as a legal sentence in English.
- But we'd like to get a parse tree back!
- Solution:
  - We can work our way back through the table and collect all the partial solutions into one parse tree.
  - Cells will need to be augmented with "backpointers", i.e. With a pointer to the cells that the current cell covers.

From recognition to parsing

	1	2	3	4	5
0	Det	NP			S
1		N			
2			V		VP
3				Det	NP
4					N

From recognition to parsing

	1	2	3	4	5
0	Det	NP			S
1		N			
2			V		VP
3				Det	NP
4					N

NB: This algorithm always fills the top “triangle” of the table!

What about ambiguity?

- The algorithm does not assume that there is only one parse tree for a sentence.
  - (Our simple grammar did not admit of any ambiguity, but this isn't realistic of course).

- There is nothing to stop it returning several parse trees.

- If there are multiple local solutions, then more than one non-terminal will be stored in a cell of the table.

CFG definition (reminder)

- A CFG is a 4-tuple: (N,Σ,R,S):
  - N = a set of non-terminal symbols (e.g. NP, VP)
  - Σ = a set of terminals (e.g. words)
    - N and Σ are disjoint (no element of N is also an element of Σ)
  - R = a set of rules of the form $A \rightarrow \beta$ where:
    - A is a non-terminal (a member of N)
    - β is any string of terminals and non-terminals
  - S = a designated start symbol (usually, “sentence”)

Motivation

- Context-free grammars can be generalized to include probabilistic information by adding it to CFG rule
- Probabilistic Context Free Grammars (PCFGs) are the simplest and most natural probabilistic model for tree structures and the algorithms for them are closely related to those for HMMs.
- PCFG are also known as Stochastic Context-Free Grammar (SCFG)

Formal Definition of a PCFG

- A PCFG consists of:
  - A set of terminals, $\{w^k\}$, $k= 1,\ldots,V$
  - A set of nonterminals, N^i, $i= 1,\ldots, n$
  - 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: $\forall i \quad \sum_j P(N^i \rightarrow \xi^j) = 1$

Probability of a Derivation Tree and a String

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

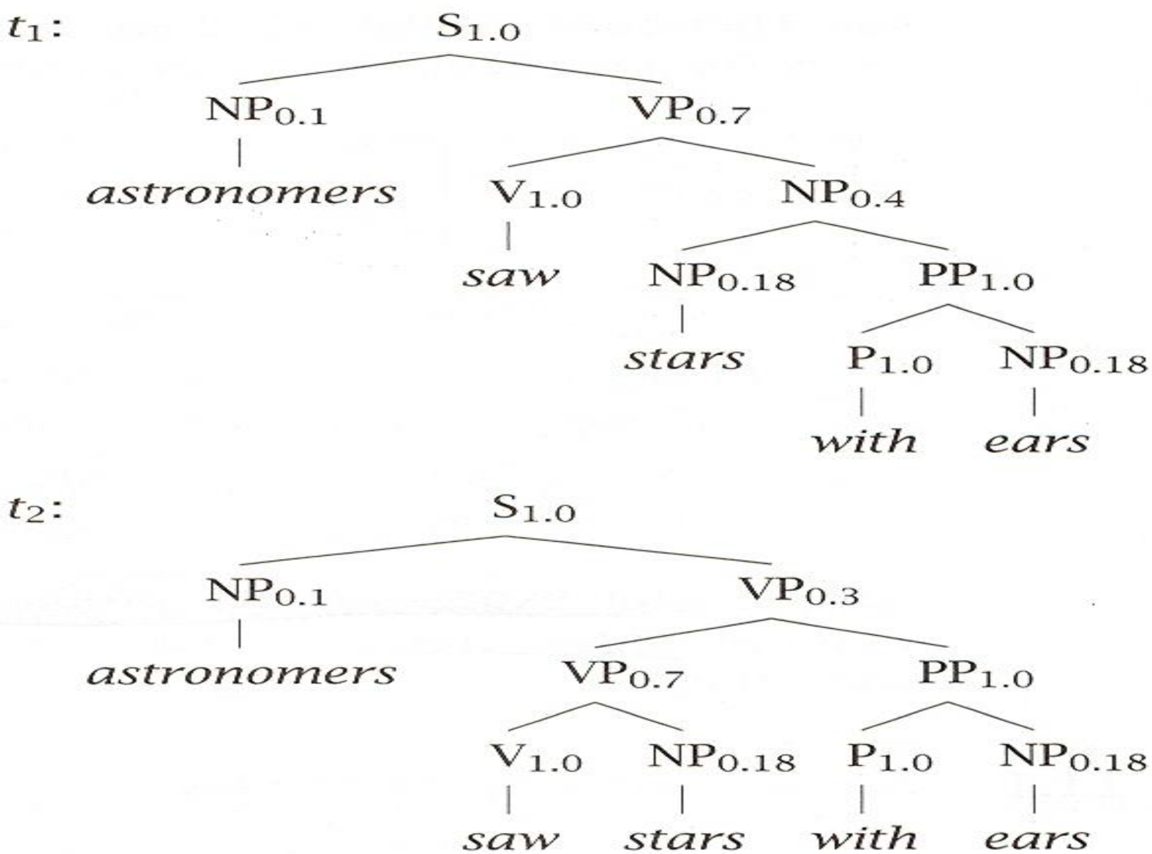
  $$P(T) = \Pi_{i=1..k}\ p(r(i))$$

  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 $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability*.
- The most likely parse tree for a sentence s is

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

PCFG based Grammar

- PCFGs augments CFGs by including a probability for each rule in the grammar.

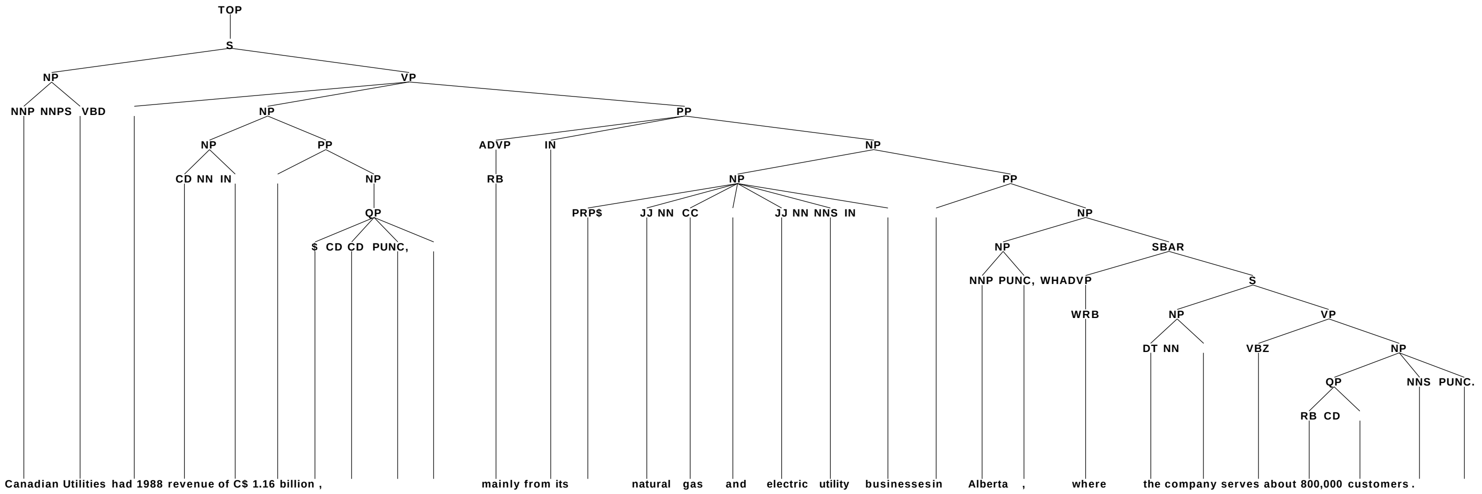
- The probability for a parse tree is the product of probabilities for the rules in the tree

- 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

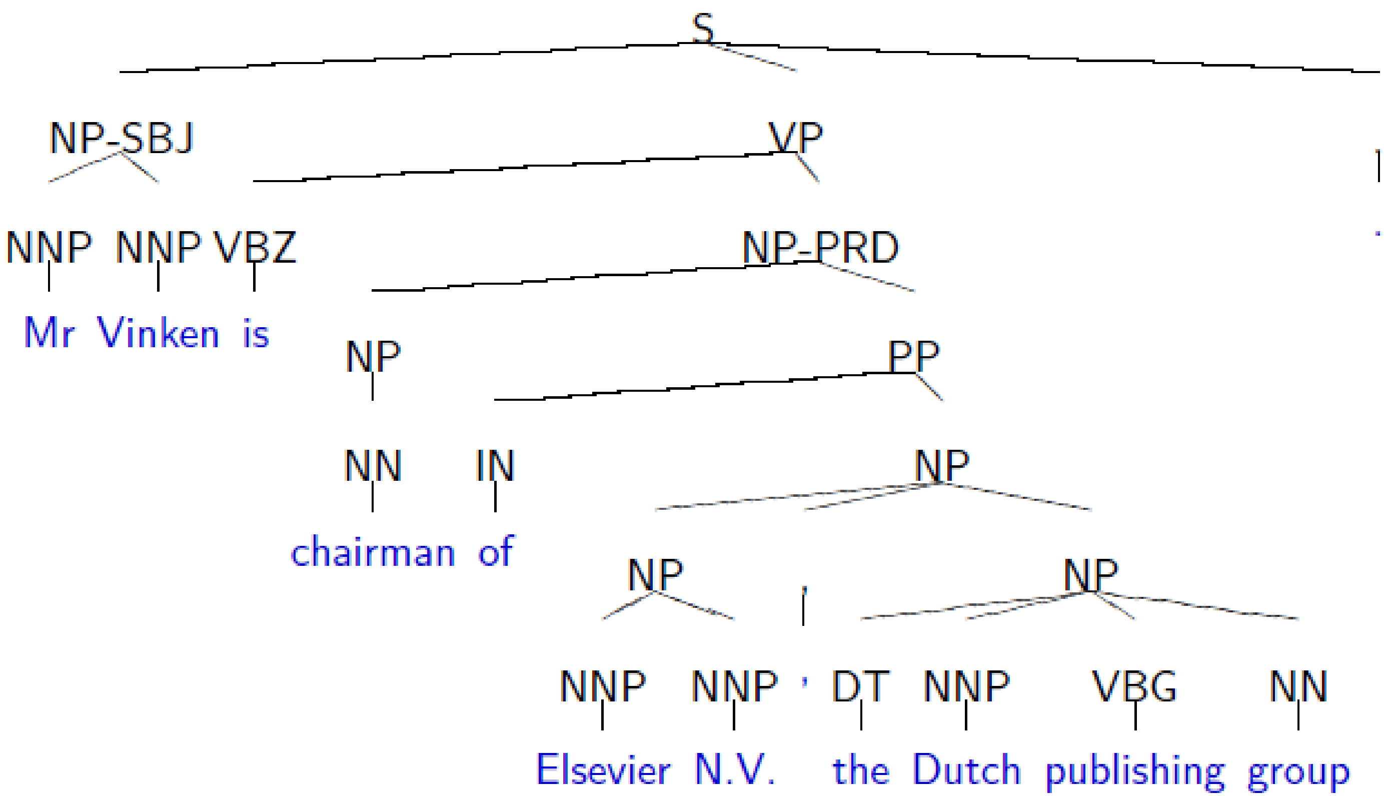
Data for Parsing Experiments: Treebanks

- 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 $S \rightarrow NP\ VP$
    - (as opposed to $S \rightarrow VP$, or $S \rightarrow NP\ VP\ PP$, 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 $A \rightarrow \beta$ must sum to 1
  - in this case, we say the PCFG is consistent

Word/Tag Counts

	N	V	ARI	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)	.360
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		

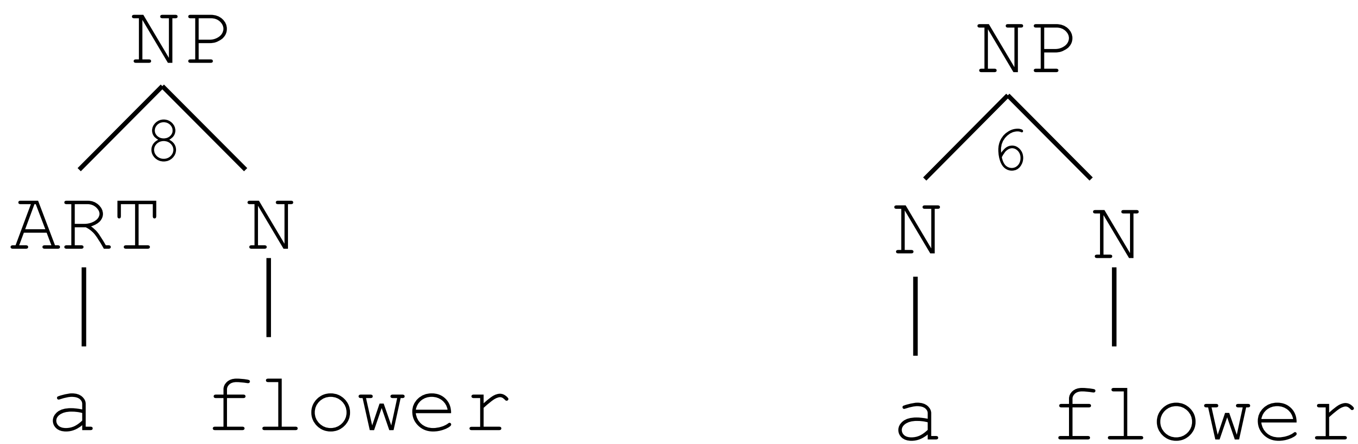
The PCFG

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

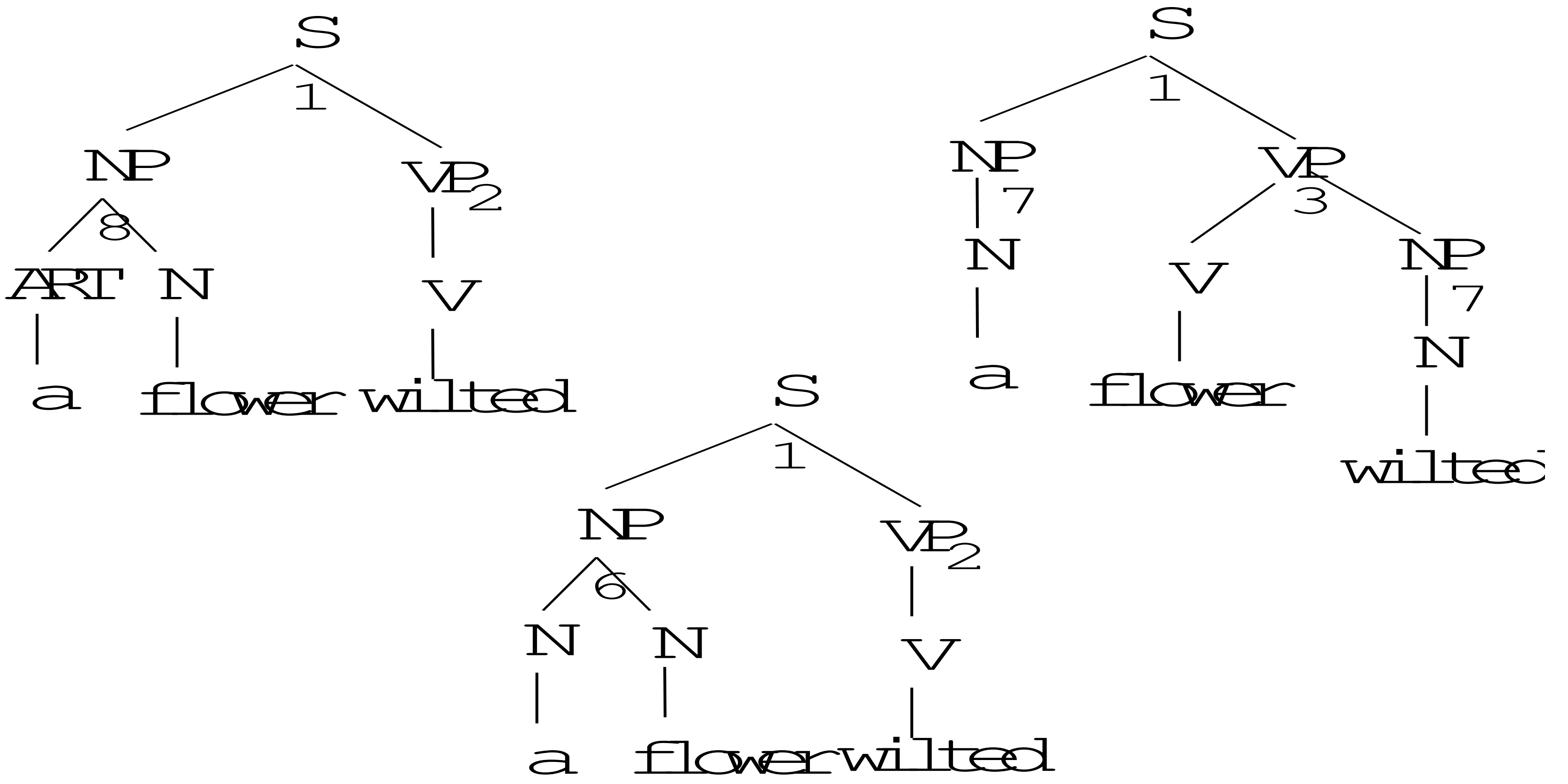
Rule	Count for LHS	Count for Rule	PROB
1. S → NP VP	300	300	1
2. VP → V	300	116	.386
3. VP → V NP	300	118	.393
4. VP → V NP PP	300	66	.22
5. NP → NP PP	1032	241	.23
6. NP → N N	1032	92	.09
7. NP → N	1032	141	.14
8. NP → ART N	1032	558	.54
9. PP → P NP	307	307	1

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.

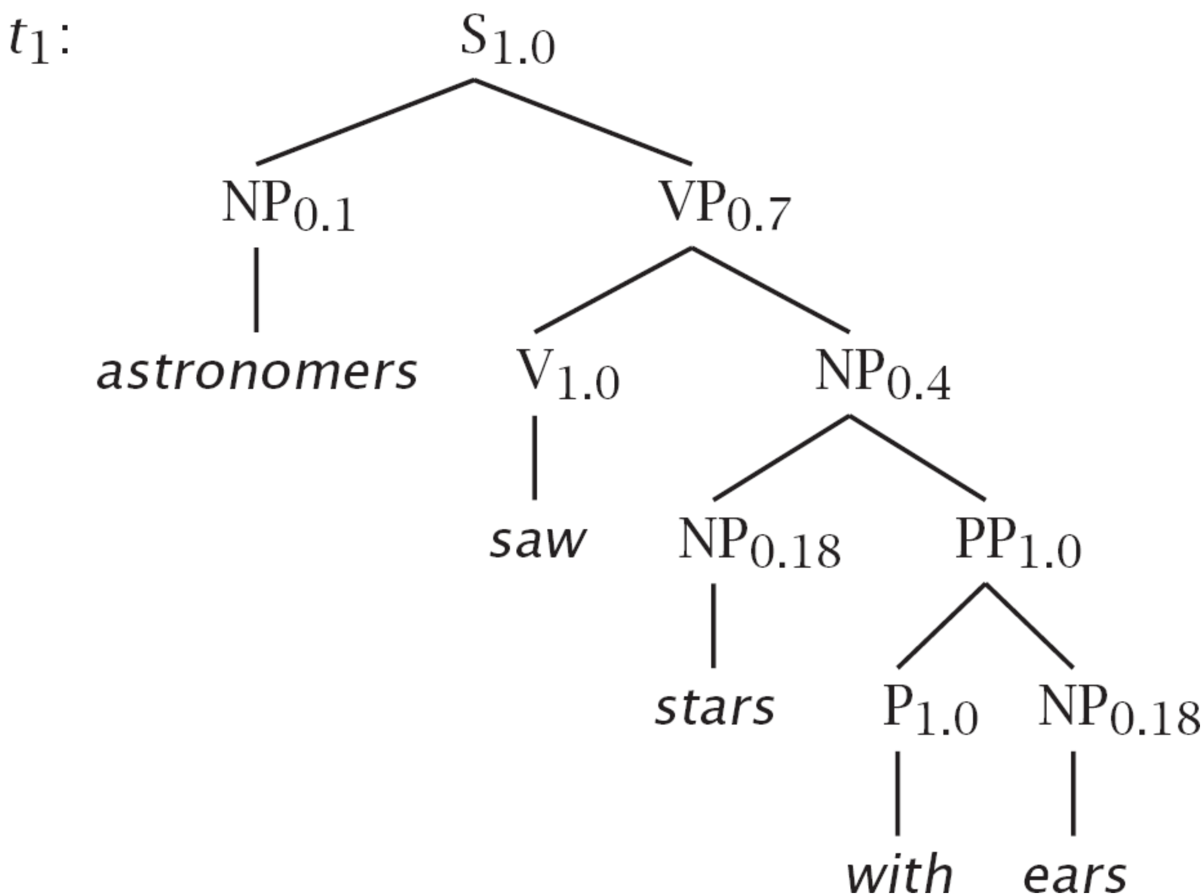
Example

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

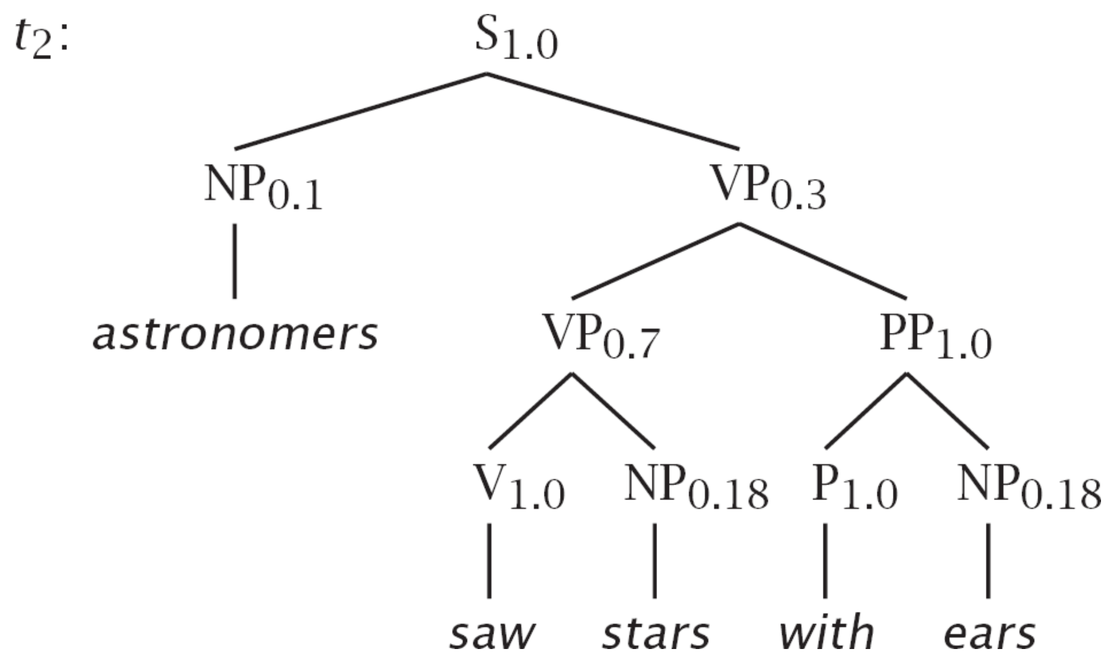
Slide based on "Foundations of Statistical Natural Language Processing" by Christopher Manning and Hinrich Schütze

astronomers saw stars with ears



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

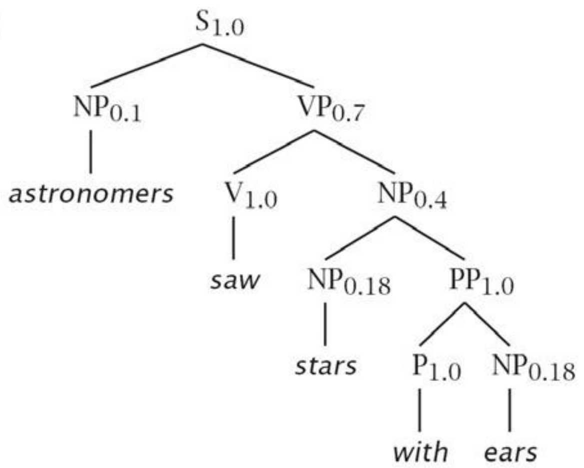
astronomers saw stars with ears



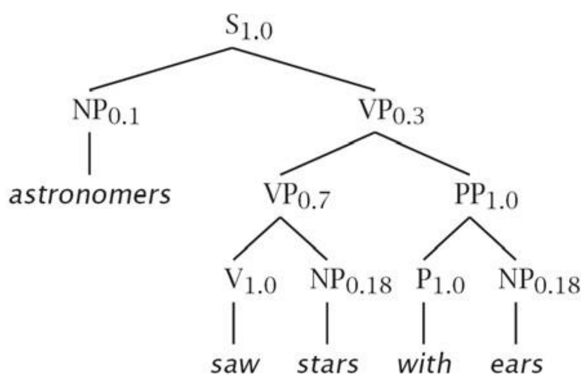
Slide based on "Foundations of Statistical Natural Language Processing" by Christopher Manning and Hinrich Schütze

Probabilities

t_1:



t_2:



$$
\begin{aligned}
P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\
&\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0009072 \\
P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\
&\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0006804 \\
P(w_{15}) &= P(t_1) + P(t_2) = 0.0015876
\end{aligned}
$$

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.

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.

Probabilistic Parsing Implementation Demo

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 Structural Context
  - How NP expands depends on position

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

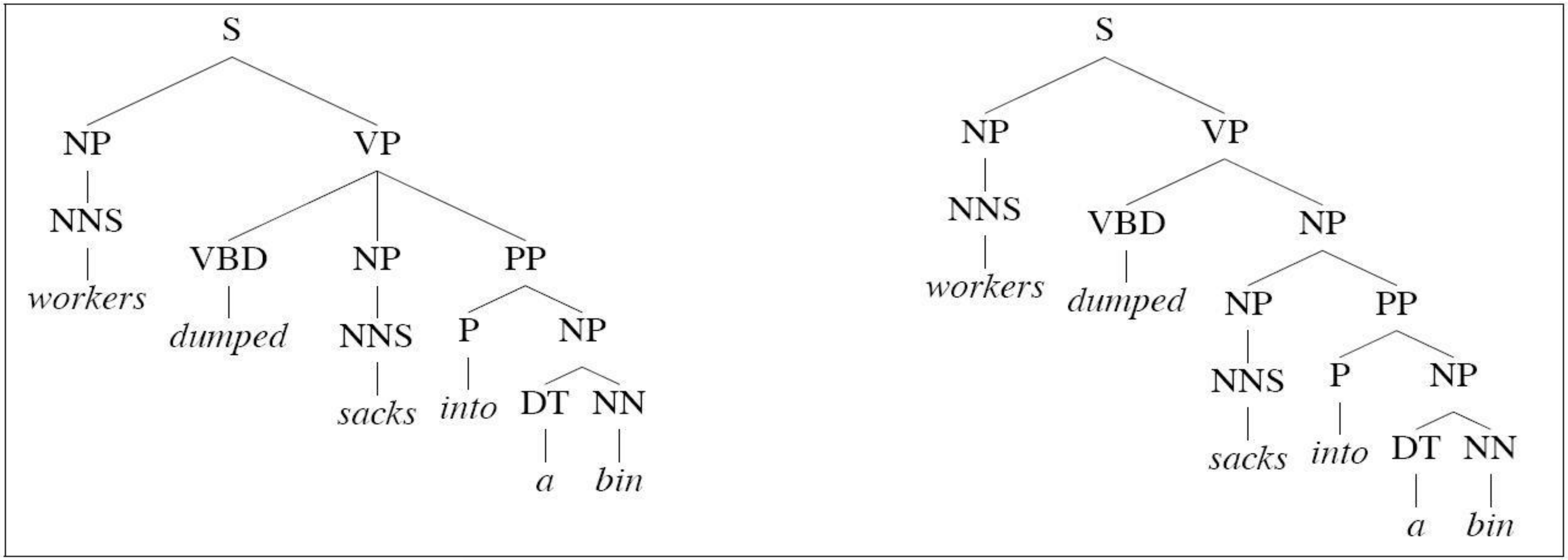
Flaws I: Structural independence

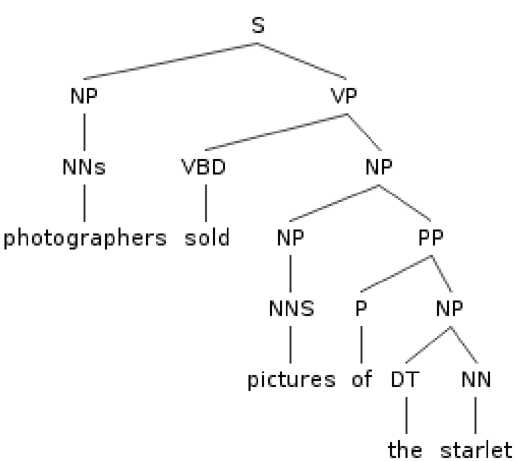
- Probability of a rule r expanding node n depends only on n.
- Independent of other non-terminals

- Example:
  - P(NP → Pro) is independent of where the NP is in the sentence
  - but we know that NP→Pro is much more likely in subject position
  - Francis et al (1999) using the Switchboard corpus:
    - 91% of subjects are pronouns;
    - only 34% of objects are pronouns

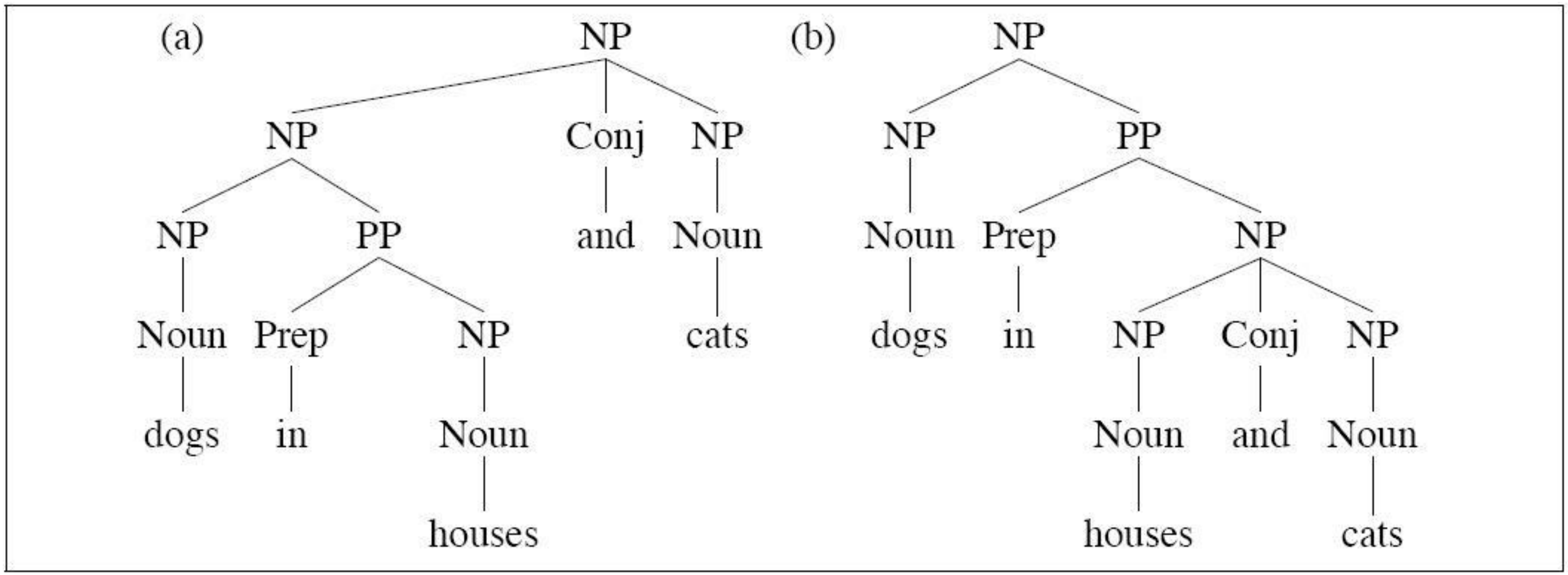
Flaws II: lexical independence

- vanilla PCFGs ignore lexical material
  - e.g. P(VP → V NP PP) independent of the head of NP or PP or lexical head V
- Examples:
  - prepositional phrase attachment preferences depend on lexical items; cf:
    - Workers dumped [sacks into a bin]
    - Workers dumped [sacks] [into a bin] (preferred parse)
  - coordination ambiguity:
    - [dogs in houses] and [cats]
    - [dogs] [in houses and cats]


S
NP VP
NNS VBD NP PP
workers dumped NNS P NP
sacks into DT NN
a bin
S
NP VP
NNS VBD NP
workers dumped NP PP
NNS P NP
sacks into DT NN
a bin


S
NP VP
NNs VBD NP
photographers sold NP PP
NNS P NP
pictures of DT NN
the starlet

Conjunction attachment



Lexicalised PCFGs

- Attempt to weaken the lexical independence assumption.
- Most common technique:
  - mark each phrasal head (N,V, etc) with the lexical material
  - this is based on the idea that the most crucial lexical dependencies are between head and dependent
  - E.g.: Charniak 1997, Collins 1999

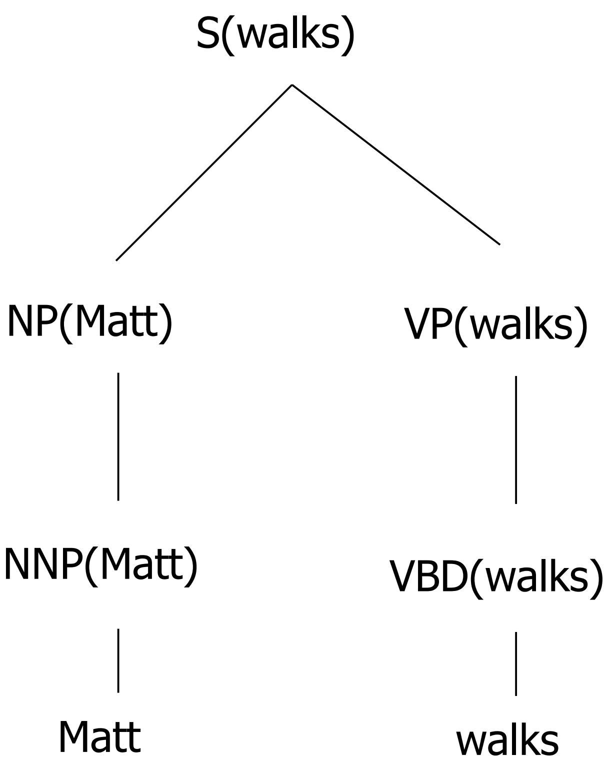
Lexicalised PCFGs: *Matt walks*

- Makes probabilities partly dependent on lexical content.

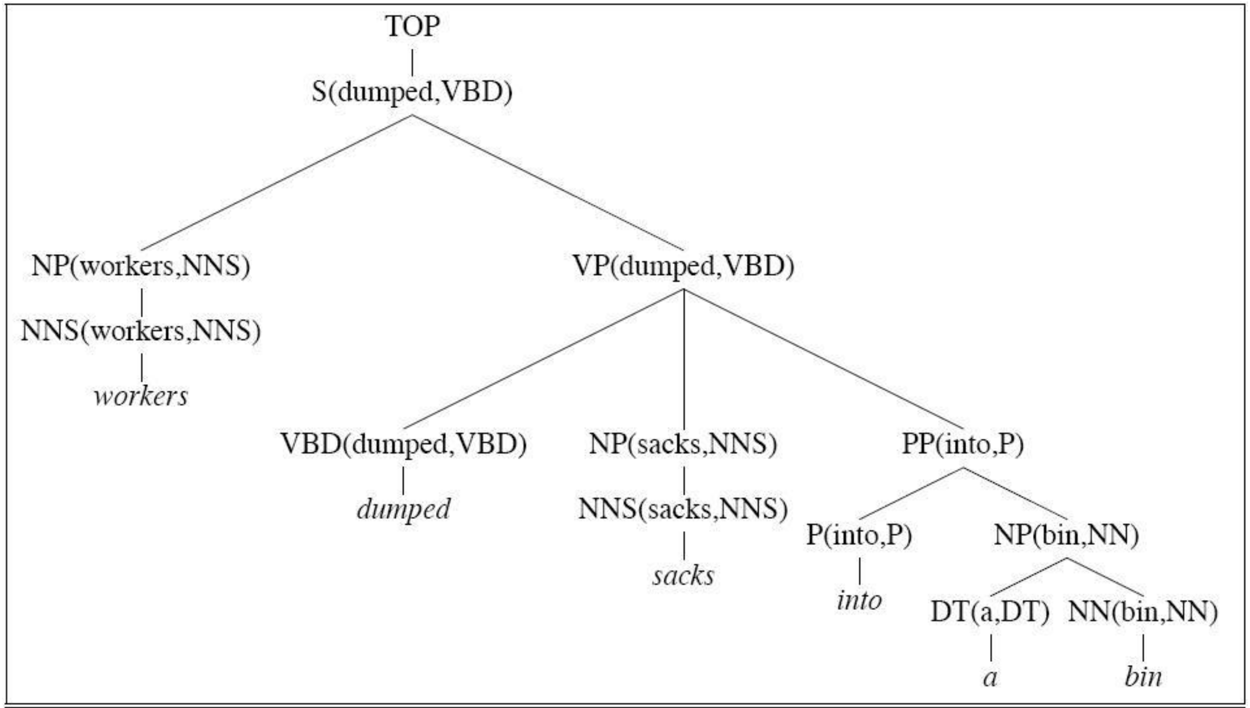
- P(VP→VBD|VP) becomes:

  P(VP→VBD|VP,h(VP)=walks)

- NB: normally, we can't assume that all heads of a phrase of category C are equally probable.



Lexical attachment



Internal Rules					**Lexical Rules**		
TOP	→	S(dumped,VBD)			NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)		VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)			NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS)	PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)		DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)		NN(bin,NN)	→	bin

Practical problems for lexicalised PCFGs

- Data sparseness: we don't necessarily see all heads of all phrasal categories often enough in the training data

- Flawed assumptions: lexical dependencies occur elsewhere, not just between head and complement
    - *I got the easier problem of the two to solve*
    - *of the two* and *to solve* are very likely because of the prehead modifier *easier*

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*?

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

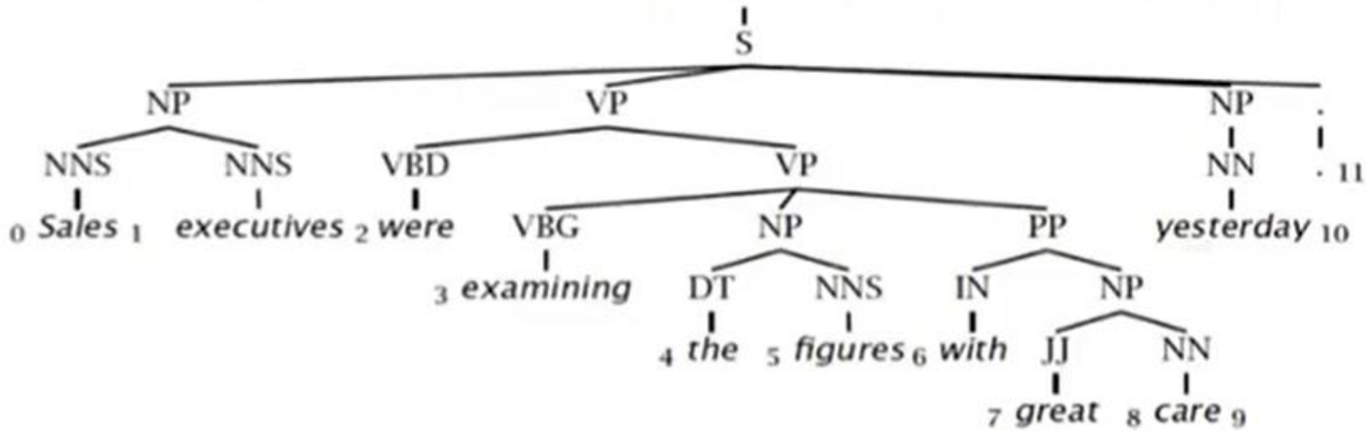
- Does *correct* mean *recall*?

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

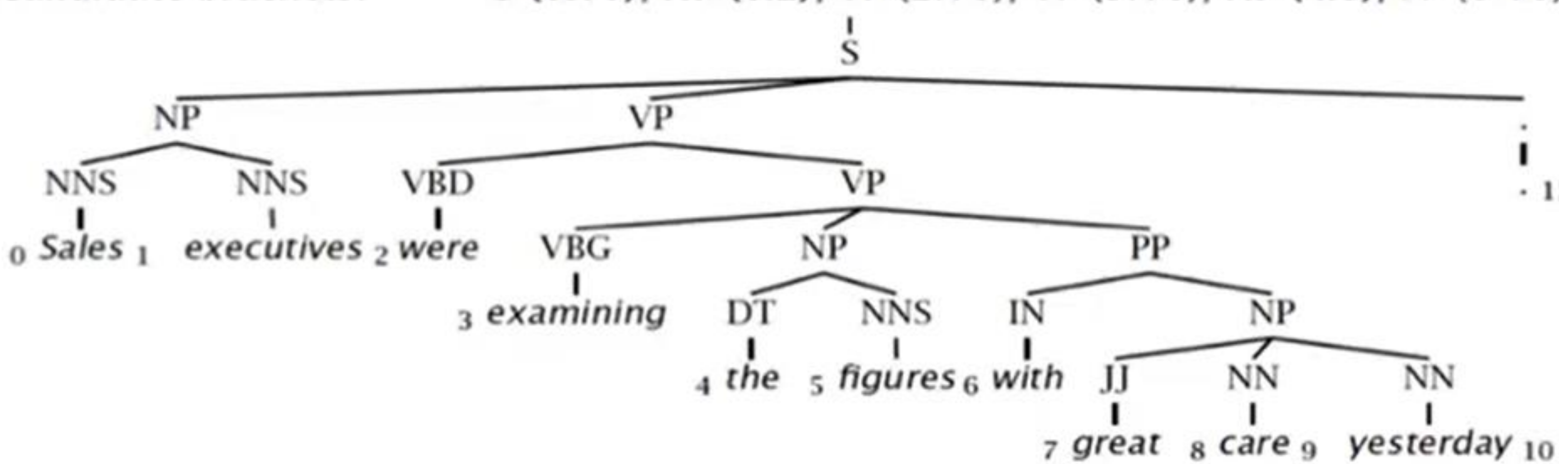
Credits: Philipp Koehn

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)

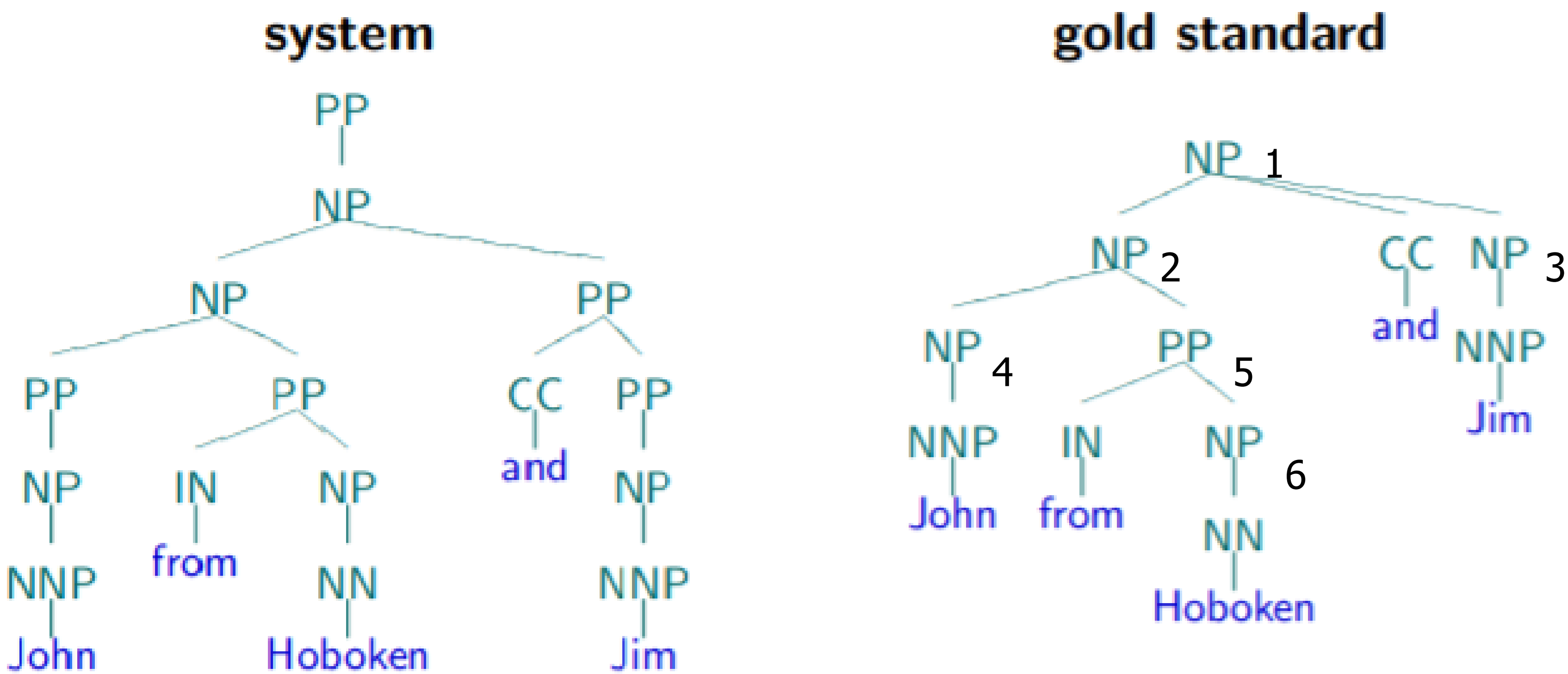
Candidate brackets: 7

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

Thank you for your time!!