Natural Language Processing

Syntactic Parsing

Alessandro Moschitti & Olga Uryupina
Department of information and communication technology
University of Trento
Email: moschitti@disi.unitn.it uryupina@gmail.com
Based on the materials by Barbara Plank

NLP: why?

Texts are objects with inherent complex structure. A simple BoW model is not good enough for text understanding.

Natural Language Processing provides models that go deeper to uncover the meaning.

- Part-of-speech tagging, NER
- Syntactic analysis
- Semantic analysis
- Discourse structure

Overview

- Linguistic theories of syntax
  - Constituency
  - Dependency
- Approaches and Resources
  - Empirical parsing
  - Treebanks
- Probabilistic Context Free Grammars
  - CFG and PCFG
  - CKY algorithm
- Evaluating Parsing
- Dependency Parsing
- State-of-the-art parsing tools

Two approaches to syntax

- **Constituency**
  - Groups of words that can be shown to act as single units: noun phrases: “a course”, “our AINLP course”, “the course usually taking place on Thursdays”,..
- **Dependency**
  - Binary relations between individual words in a sentence: “missed ➔ I”, “missed ➔ course”, “course ➔the”, “course ➔on”, “on ➔Friday”.

Constituency (phrase structure)

- Phrase structure organizes words into nested constituents
- What is a constituent? (Note: linguists disagree..)
  - Distribution:

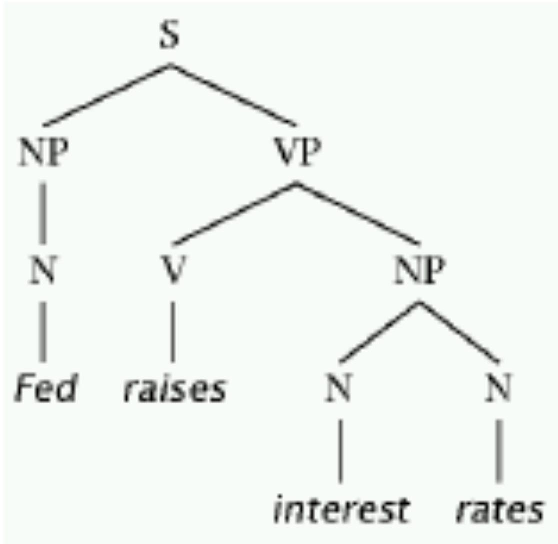
    I’m attending the AINLP course.

    The AINLP course is on Thursday.
  - Substitution/expansion

    I’m attending the AINLP course.

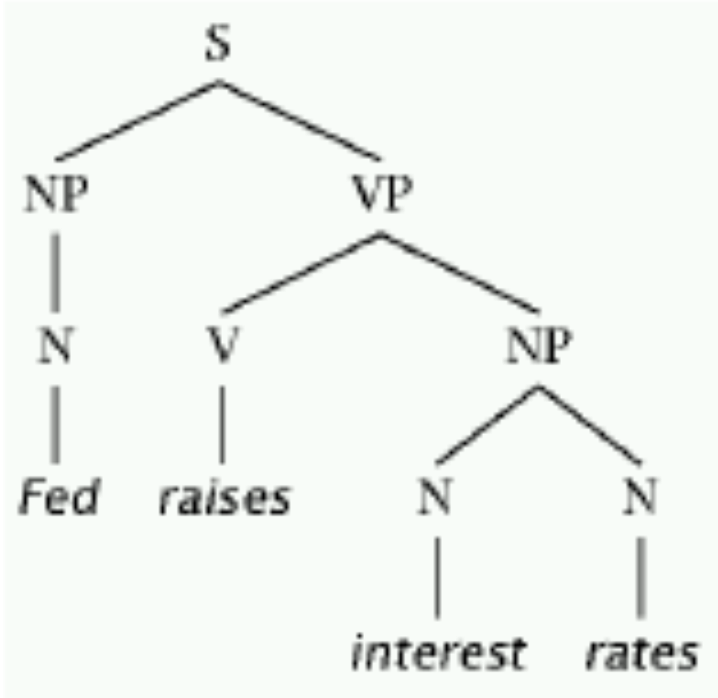
    I’m attending it.

    I’m attending the course of Prof. Moschitti.



Bracket notation of a tree

(S (NP (N Fed)) (VP (V raises) (NP (N interest) (N rates)))



Grammars

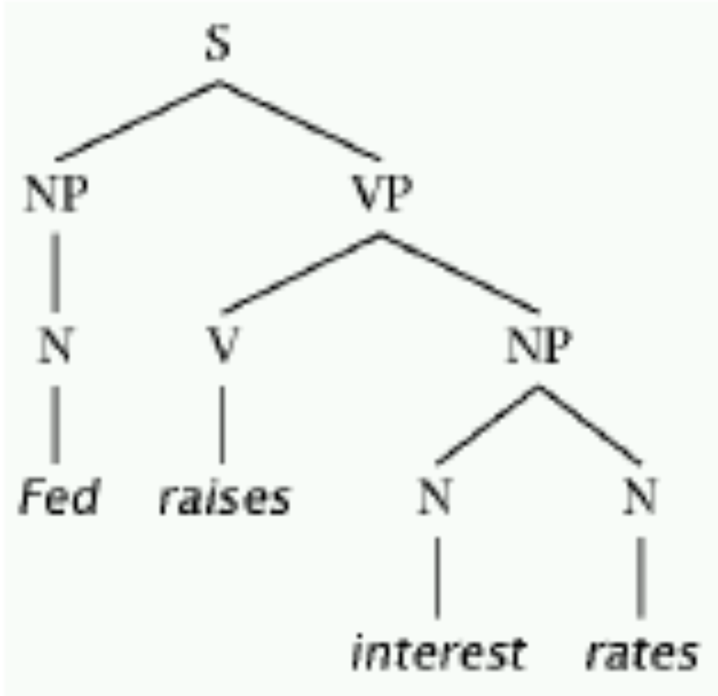
A grammar models possible constituency structures:

S ➔ NP VP

NP ➔ N

NP ➔ N N

VP ➔ V NP



Headed phrase structure

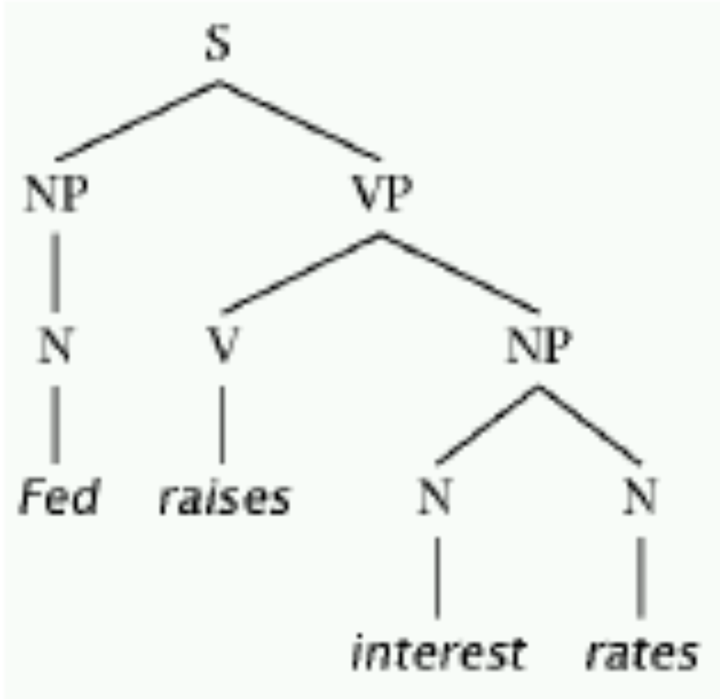
Each constituent has a head:

S → NP VP*

NP → N*

NP → N N*

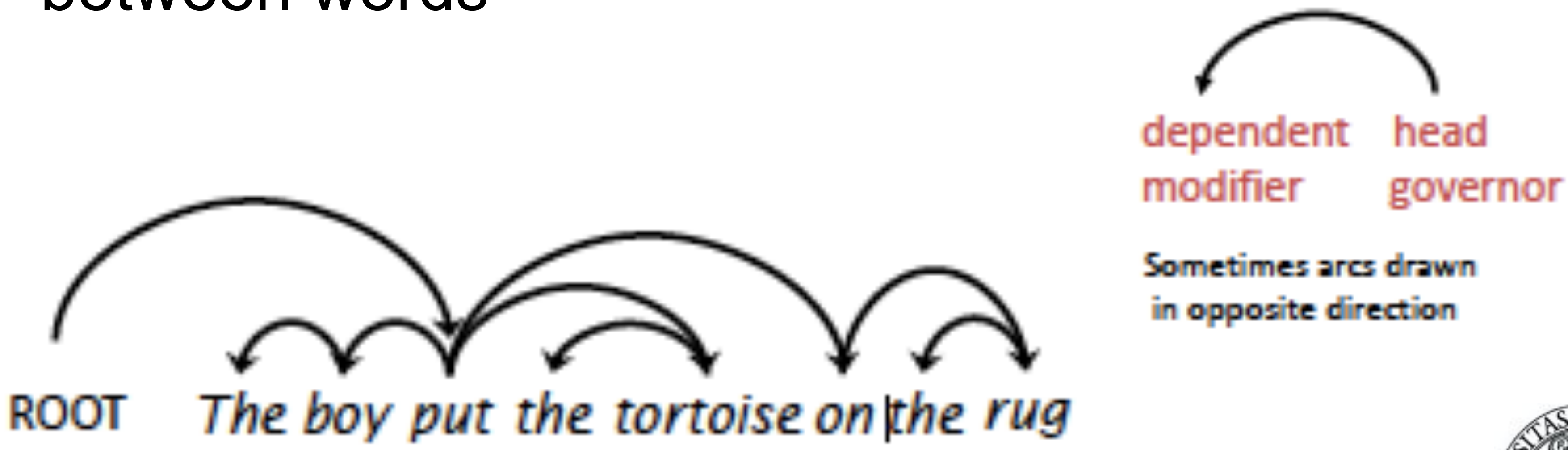
VP → V* NP



Dependency structure

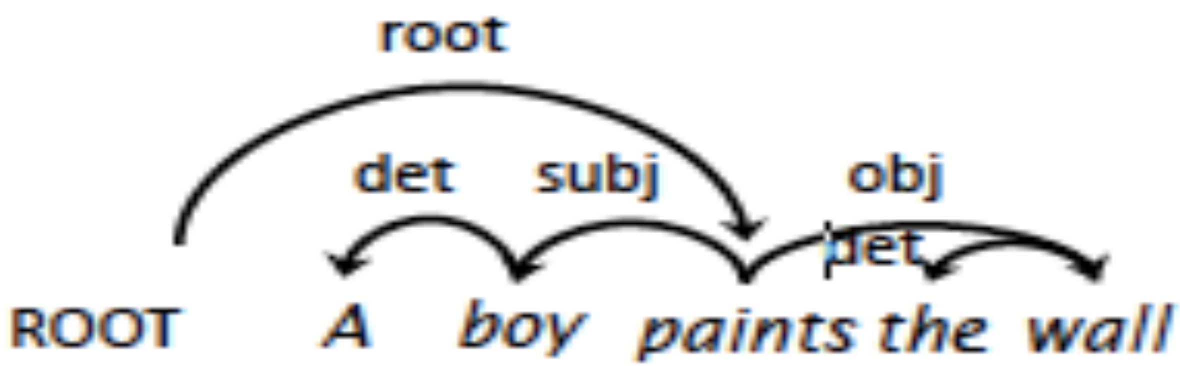
A dependency parse tree is a tree structure where:

- the nodes are words,
- the edges represent syntactic dependencies between words



Dependency labels

- Argument dependencies:
  - subject (subj), object (obj), indirect object (iobj)
- Modifier dependencies:
  - determiner (det), noun modifier (nmod), etc



Dependency vs. Constituency

Dependency structure explicitly represents

- head-dependent relations (directed arc),
- functional categories (arc lables).

Constituency structure explicitly represents

- phrases (non-terminal nodes),
- structural categories (non-terminal labels)
- possibly some functional categories (grammatical functions, e.g. PP-LOC)

Dependencies are better for free word order languages

It's possible to convert dependencies to constituencies and vice versa with some effort

Hybrid approaches (e.g. Dutch Alpino grammar)

Parsing algorithms

Classical (pre-1990) NLP parsing

- Symbolic **grammars** + lexicons
  - CFG (context-free grammars)
  - richer grammars (model context dependencies, computationally prohibitively expensive)
- Use grammars and proof systems to **prove** parses from words
- Problems: doesn't scale, poor coverage

Grammars again

Grammar

S ➔ NP VP

NP ➔ N

NP ➔ N N

VP ➔ V NP

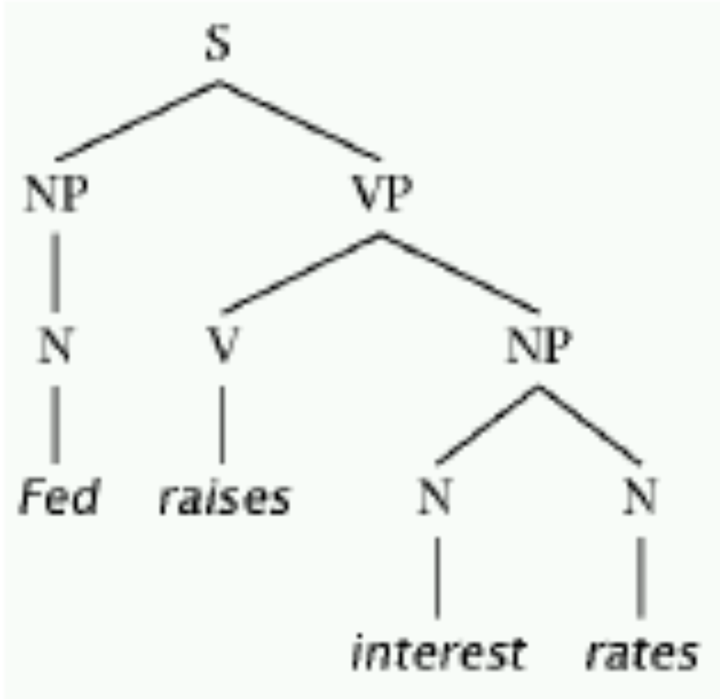
Lexicon

N ➔ Fed

N ➔ interest

N ➔ rates

V ➔ raises



Problems with Classical Parsing

- CFG -- unlikely/weird parses
  - can be eliminated through (categorial etc) constraints,
  - but the attempt makes the grammars not robust

  ➔ In traditional systems, around 30% of sentences have no parse
- A less constrained grammar can parse more sentences
  - But it produces too many alternatives with no way to chose between them

Statistical parsing allows to find the most probable parse for any sentence

Treebanks

The Penn Treebank (Marcus et al. 1993, CL)

- 1M words from the 1987-1989 Wall Street Journal newspaper

Many other projects since then

Torino Tree Bank (TUT) for Italian

((S (NP-SBJ (DT The) (NN move)) (VP (VBD followed) (NP (NP (DT a) (NN round)) (PP (IN of) (NP <..>)) (. .))

Treebanks: why?

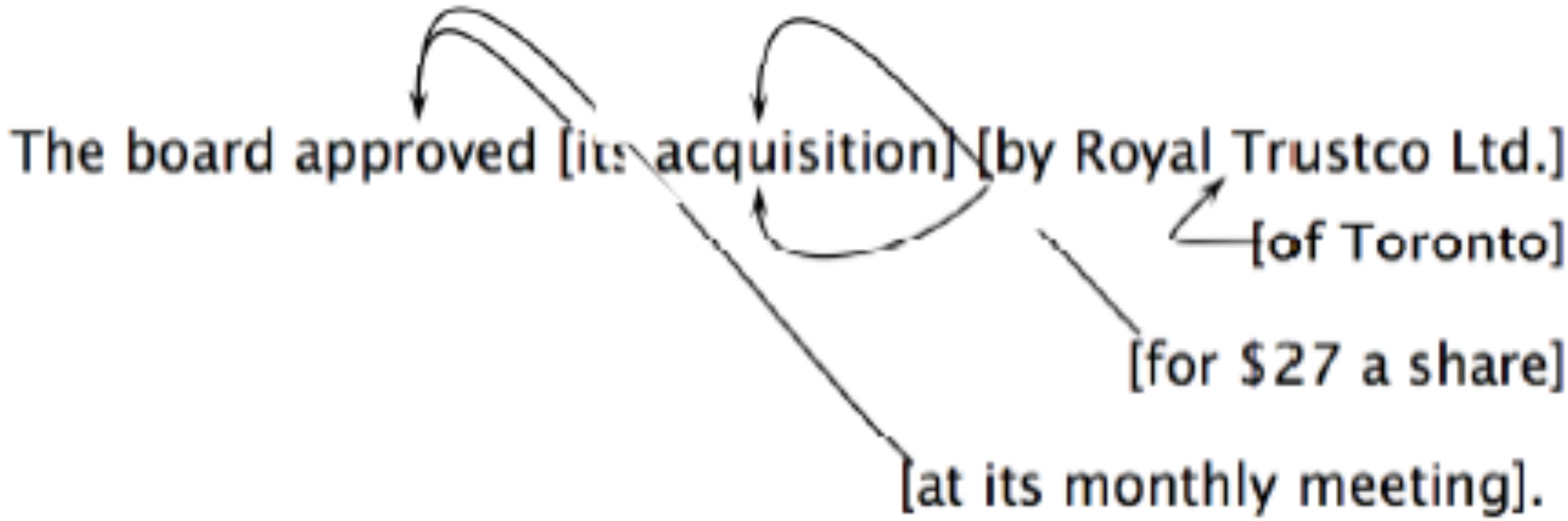
Building a treebank seems slower and less useful since it cannot parse anything, unlike grammars..

But in reality, a treebank is an extremely valuable resource:

- Reusability of the labor
  - Train parsers, POS taggers, etc
  - Linguistic analysis
- Broad coverage, realistic data
- Statistics for building parsers
- A reliable way to evaluate systems

Statistical parsing: attachment ambiguities

The key parsing decision: how we "attach" various constituents?



Counting attachment ambiguities

How many distinct parses does this sentence have due to PP attachment ambiguities?

John wrote the book with a pen in the room.

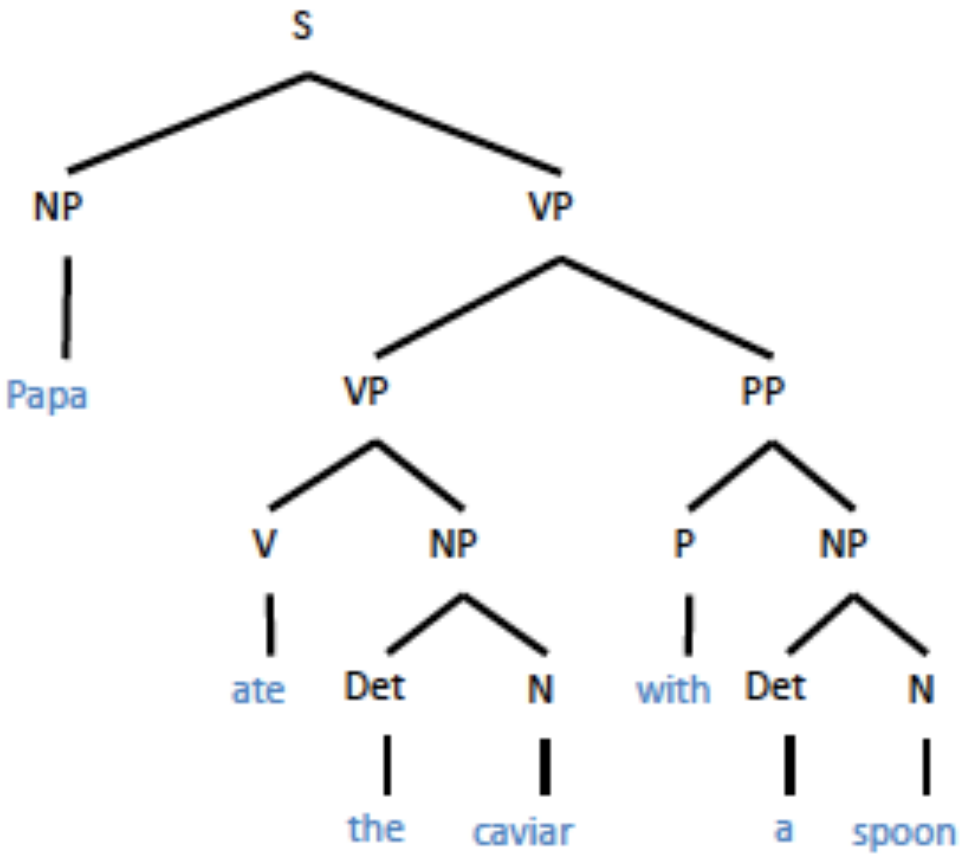
John wrote [the book] [with a pen] [in the room].
John wrote [[the book] [with a pen]] [in the room].
John wrote [the book] [[with a pen] [in the room]].
John wrote [[the book] [[with a pen] [in the room]]].
John wrote [[[the book] [with a pen]] [in the room]].

Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$ - an exponentially growing series

1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430

Ambiguity: choosing the correct parse

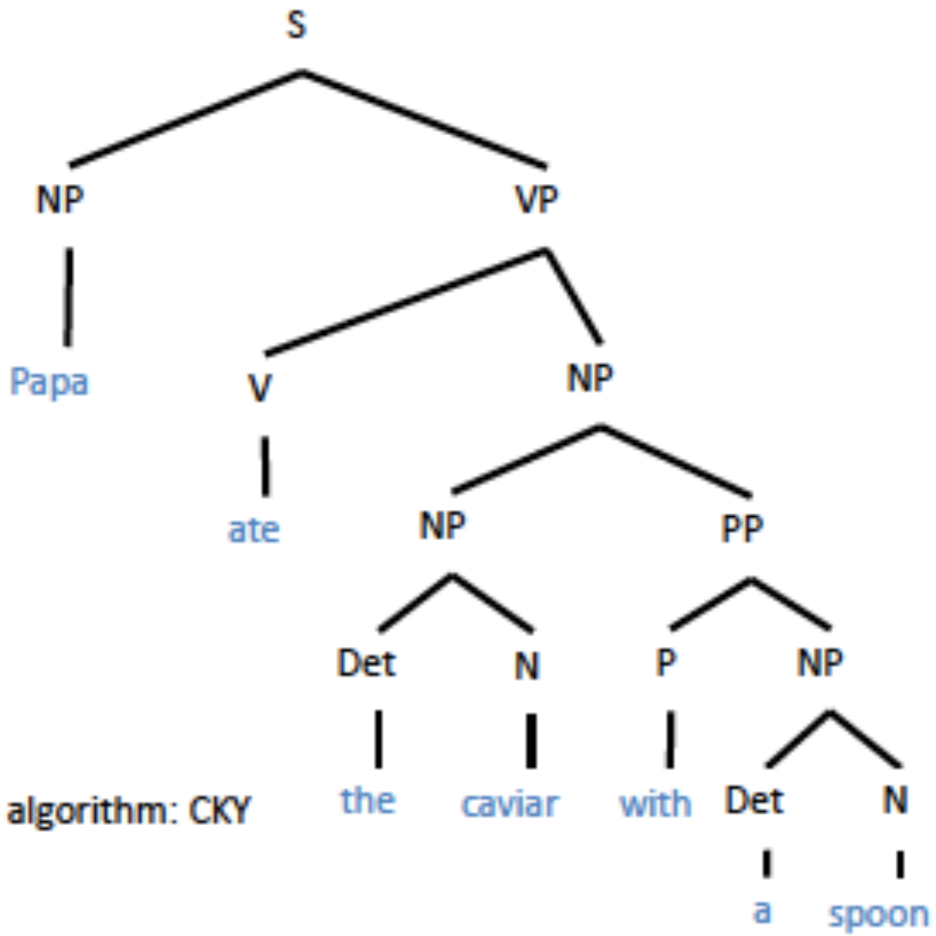
S → NP VP
NP → Det N
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP



NP → Papa
N → caviar
N → spoon
V → spoon
V → ate
P → with
Det → the
Det → a

Ambiguity: choosing the correct parse

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



NP → Papa
N → caviar
N → spoon
V → spoon
V → ate
P → with
Det → the
Det → a

→ need an efficient algorithm: CKY

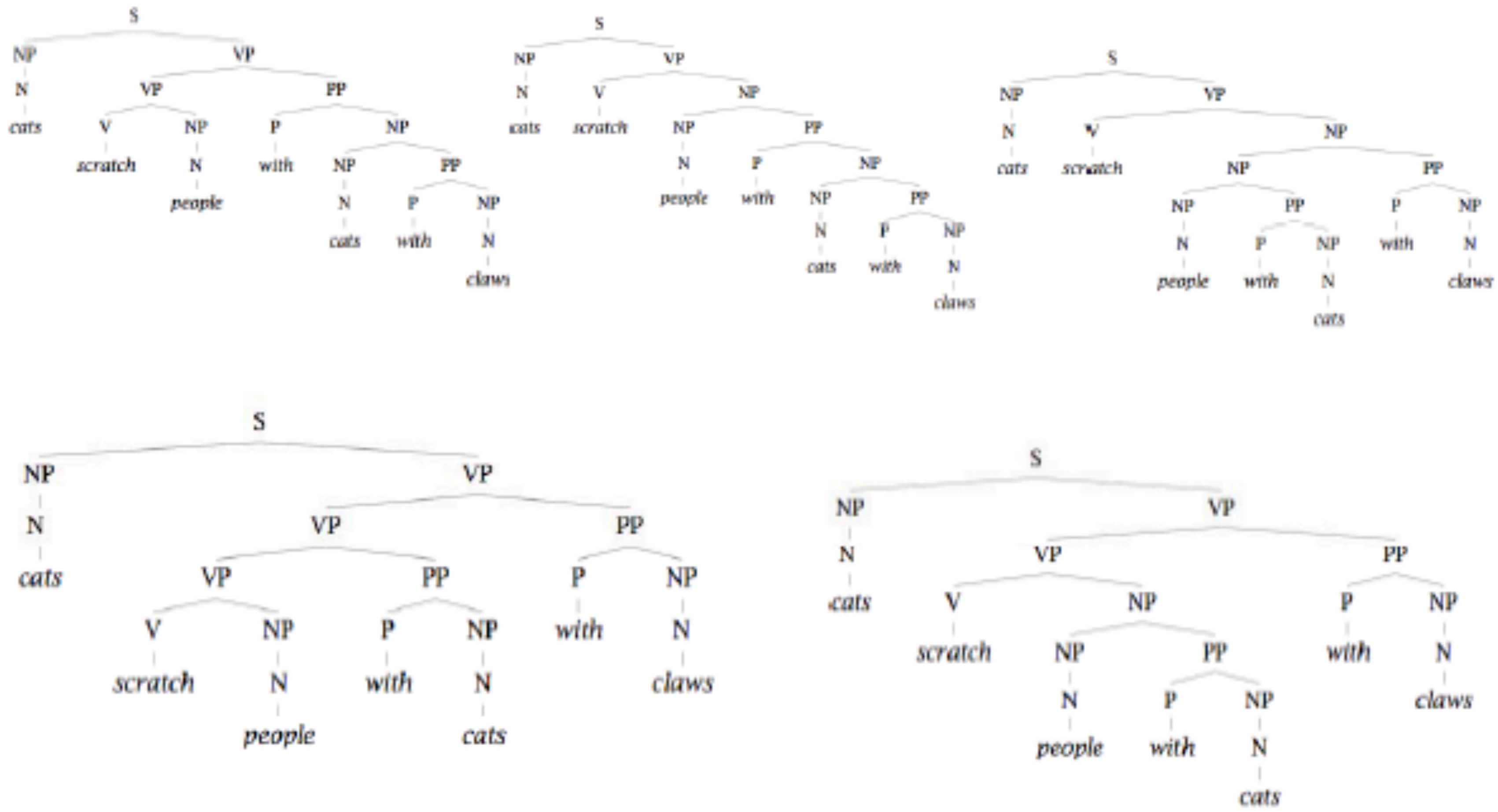
Avoiding repeated work

Parsing involves generating and testing many hypotheses, with considerable overlap. Once we've build some good partial parse, we might want to re-use it for other hypotheses.

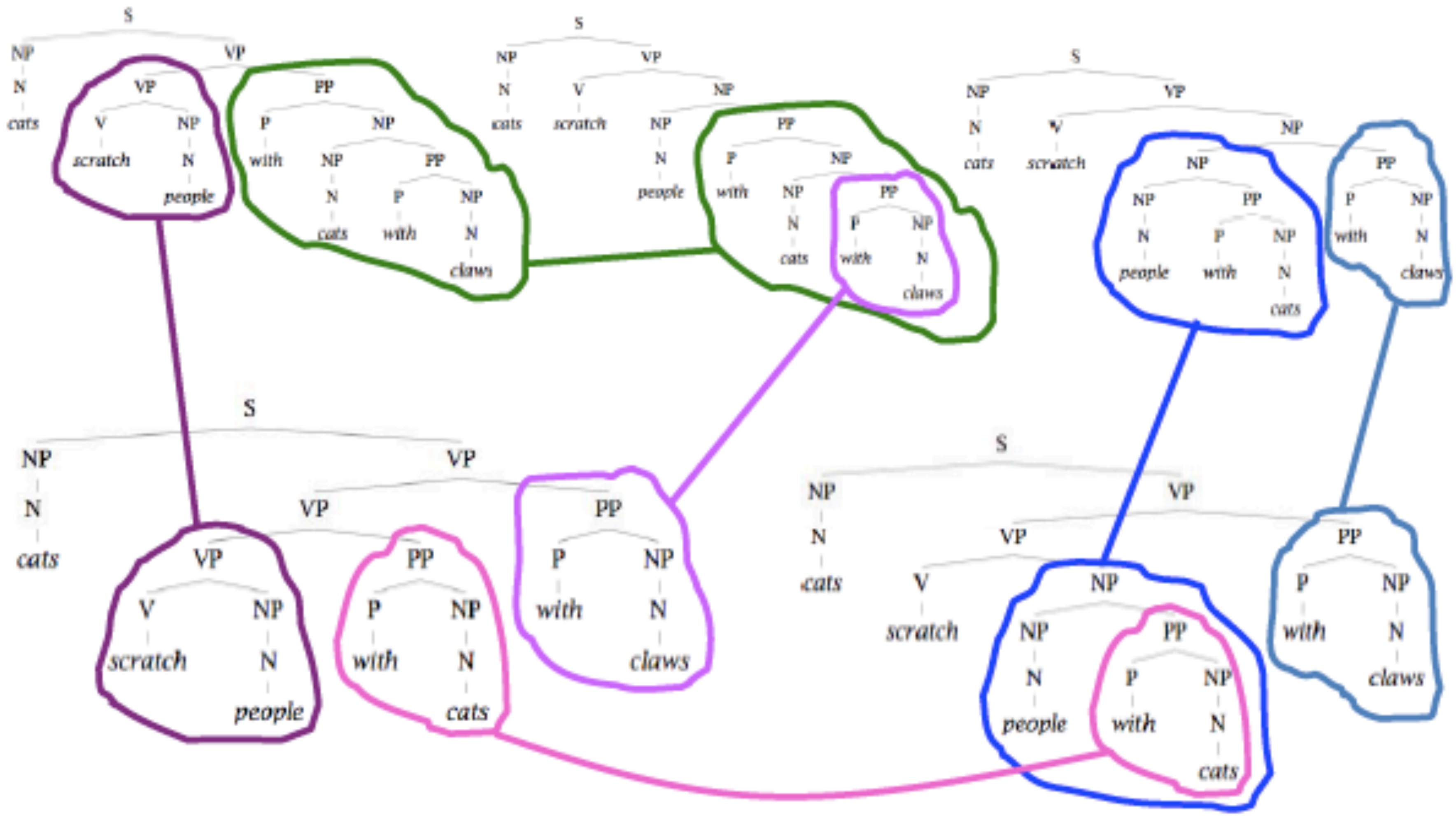
Example: Cats scratch people with cats with claws.

Avoiding repeated work

-



Avoiding repeated work



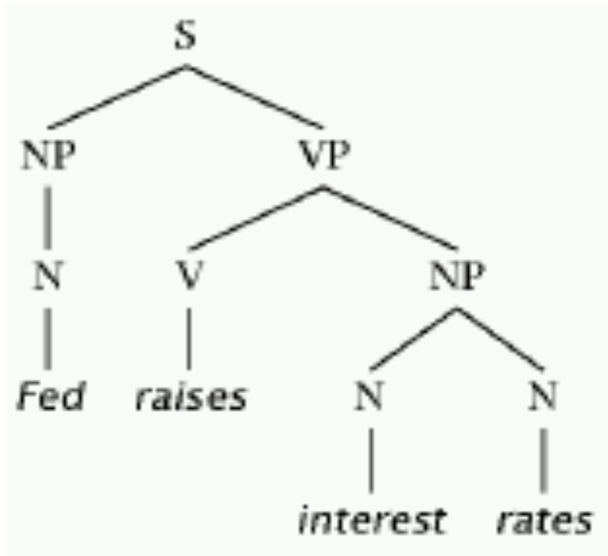
CFG and PCFG

CFG Grammar

S ➔	NP VP	(binary)
NP ➔	N	(unary)
NP ➔	N N	
VP ➔	V NP	
VP ➔	V NP PP	n-ary (n=3)

Lexicon

N ➔	Fed
N ➔	interest
N ➔	rates
N ➔	raises
V ➔	raises
V ➔	rates



Alternative parse: [Fed raises] interest [rates]

Context-Free Grammars (CFG)

G= <T,N,S,R>

T: set of terminal symbols

N: set of non-terminal symbols

S: starting symbol (“root”)

R: set of production rules X $\rightarrow \gamma$

- $X \in N$, $\gamma \in N \cup T$

A grammar G generates a language L.

Probabilistic (Stochastic) Context-Free Grammars – PCFG

G= <T,N,S,R,P>

T: set of terminal symbols

N: set of non-terminal symbols

S: starting symbol ("root")

R: set of production rules X ➔γ

P: a probability function 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: for each sentence, it generates a probabilistic distribution of parses

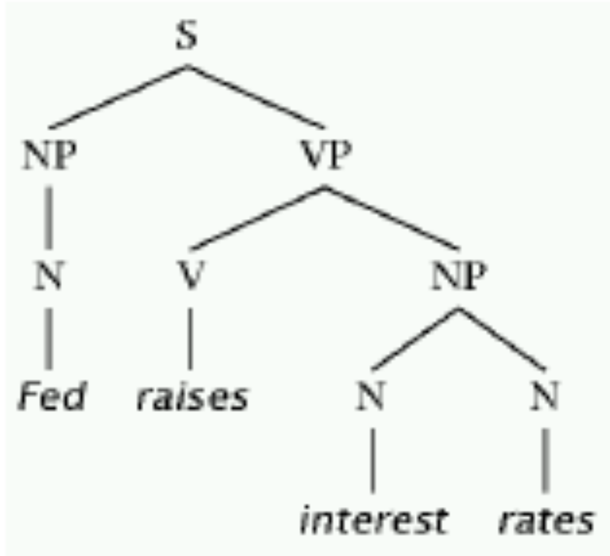
CFG and PCFG

PCFG Grammar

S ➔	NP VP	1.0
NP ➔	N	0.3
NP ➔	N N	0.7
VP ➔	V NP	0.9
VP ➔	V NP PP	0.1

Lexicon

N ➔	Fed	0.5
N ➔	interest	0.2
N ➔	rates	0.1
N ➔	raises	0.2
V ➔	raises	0.7
V ➔	rates	0.3



Alternative parse: [Fed raises] interest [rates]

Getting PCFG probabilities

- Get a large collection of parsed sentences (treebanks!)
- Collect counts for each production rules
- Normalize per X
- Done!

Counting probabilities of trees and strings

P(t) – the probability of a tree *t* is the product of the probabilities of all the production rules of *t*.

P(s) – the probability of the string *s* is the sum of the probabilities of the trees that yield *s.*

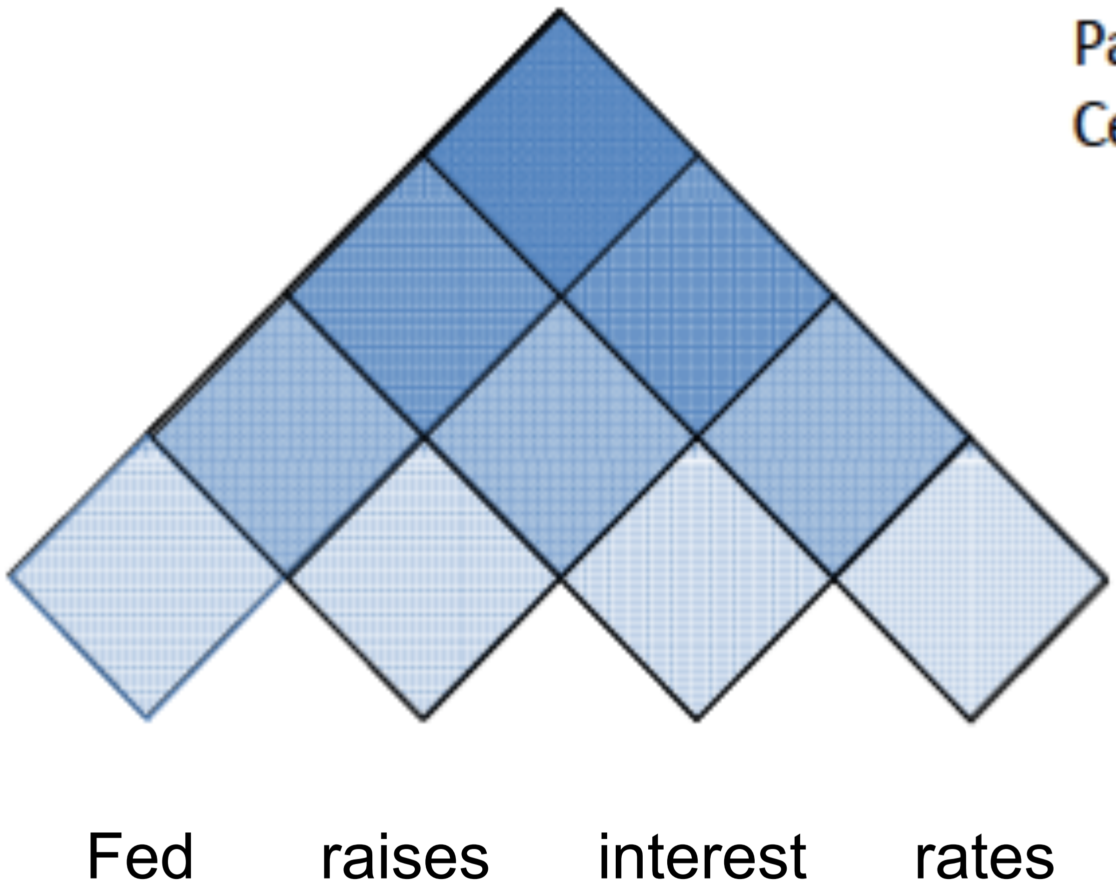
Where do we stand?

- We can choose better parses according to a PCFG grammar
  - Compute and compare tree probabilities based on the individual probabilities of PCFG production rules
- But we still do not know how to generate parse candidate efficiently
  - Exponential number of possible trees

Cocke-Kasami-Younger Parsing (CKY)

- Bottom-up parsing (starts from words)
- Use dynamic programming to avoid repeated work
- Operates on PCFGs transformed into the Chomsky Normal Form (only binary and unary production rules)
- Worst-time complexity: $O(n^3|G|)$
- Average-time complexity is better for more advanced algorithms

CKY: parsing chart



Parsing chart
Cells over spans of words

Filling the CKY chart

Objective: for each cell (== sequence of words), find its best parse for each category, with probability

How to compute the best part for a cell spanning from word *i* to word *j*?

- Generate a split: *<l,k> <k+1,j>*
- Check cells for <l,k> and for <k+1,j> -- they should contain the best parses
- Check production rules to find out how the best parses can be combined

Filling the CKY chart

Objective: for each cell (== sequence of words), find its best parse, with probability

- Start with 1-word cells (lexicon probabilities)
- Fill all 1-word cells
- Proceed with 2-word cells, then 3-word cells etc

CKY parsing: example with CFG

Fed	N				
raises		V N			
interest			V N		
rates				V N	

CKY parsing: example with CFG

Fed	N	N NP			
raises		V N	V N NP		
interest			V N	V N NP VP	
rates				V N	V N NP VP

CKY parsing: example with CFG

Fed	N	N NP	NP		
raises		V N	V N NP	NP VP	
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

CKY parsing: example with CFG

Fed	N	N NP	NP	NP	
raises		V N	V N NP	NP VP	VP NP
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

CKY parsing: example with CFG

Fed	N	N NP	NP	NP VP	?
raises		V N	V N NP	NP VP	VP NP
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

[Fed] [raises interest rates]

Fed	N	N NP	NP	NP	S
raises		V N	V N NP	NP VP	VP NP
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

[Fed raises] [interest rates]

Fed	N	N NP	NP	NP	S
raises		V N	V N NP	NP VP	VP NP
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

[Fed raises interest] [rates]

Fed	N	N NP	NP	NP VP	S
raises		V N	V N NP	NP VP	VP NP
interest			V N	V N NP VP	NP VP
rates				V N	V N NP VP

CKY for PCFG: Viterbi decoding

For each symbol in each cell, only choose the parse with the highest probability

How good are PCFG parsers?

Straightforward PCFG on Penn Treebank: 73% F

Main issue: strong independence assumption (context free grammars). This helps reduce the complexity, but it also introduces errors:

- Agreement

  e.g., “S->NP VP”, no constraint to prevent parses with singular NP and plural VP

- Subcategorization

Agreement

NP → DET N

DET → This

DET → These

N → cat

N → cats

This grammar overgenerates: it allows for phrases “this cat”, “these cats”, but also for “this cats” and “these cat”.

Subcategorization

Possible expansions might differ for different words:

Sneeze: John sneezed

Find: Please find a flight to NY

Give: Give me a cheaper fare

Help: Can you help me with a flight?

<..>

VP ➔ V, VP ➔ V NP PP, VP ➔ V NP NP

*John sneezed me with a cheaper fare

*Give with a flight

Agreement/Subcategorization: solutions

- Within (P)CFG: create more specific labels

Old rule: NP ➔ DET N

New rules: NP-sg ➔ DET-sg N-sg,

NP-pl ➔ DET-pl N-pl

Agreement/Subcategorization: solutions

Create more specific labels

+ stays within the power of CFG (==efficient)

- Ugly
- Scalability issues: too many rules, too many phenomena due to no lexicalization in the vanilla PCFG

More issues..

- Attachment ambiguity

    I'm eating sushi with tuna
    I'm eating sushi with friends

Problem: lexical items (words) are only used at a very low level and cannot help the parser to make good decisions.

Solution: head-lexicalized PCFG, more expressive grammar formalisms (HPSG, TAG,..)

Lexicalized PCFG: 88% on Penn Treebank

Head-lexicalized PCFG

Publicly available SOTA parsers: Charniak, Collins

Main idea: each constituent has a head. The head is a good representation of the phrase's structure and meaning. So, we can propagate the heads all the way up the tree.

Old rule: NP ➔ DET N

New rules: NP-cat ➔ DET-cat N*-cat

Use smoothing to correctly estimate probabilities

Example – Charniak parser: 2-stage algorithm

- Lexicalized PCFG generates n-best parses
- MaxEnt choses the best one

Dependency parsing

Dependency structure:

- nodes correspond to words
- edges/arcs correspond to relations

Properties of the dependency graph:

- connected
- acyclic
- single-head constraint for all nodes except for root

Dependency parsing

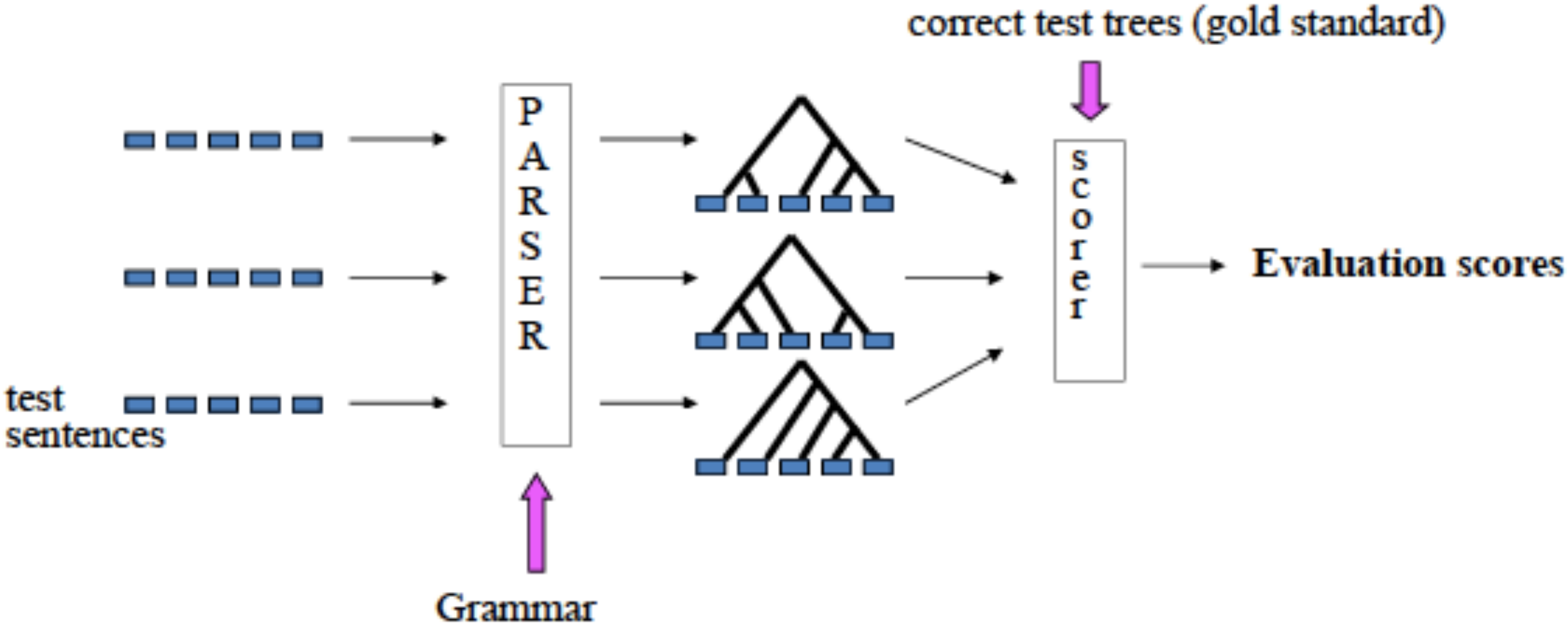
Projective vs. non-projective structures:

- non-projective structures cannot be represented without intersecting edges
  - Long-distance dependencies
  - Free word order languages
- Modern SOTA parsers can produce non-projective structures as well

Algorithms for dependency parsing

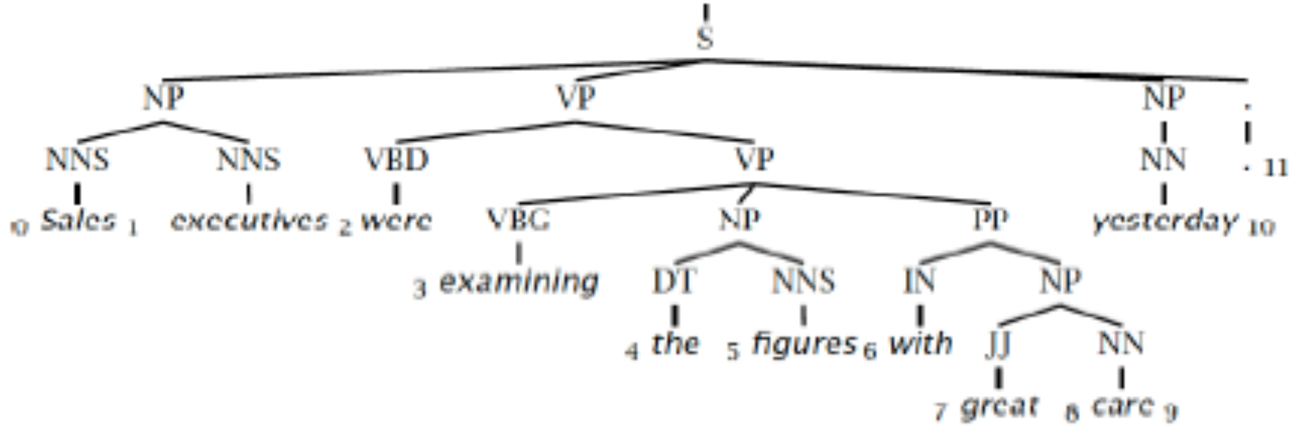
- Dynamic programming: efficiently search a space of trees to optimize some criterion
  - Dependencies as constituents (CKY-style) – Eisner
  - Sum of edge scores – Maximum Spaning Treee – MST, Bohnet
- Deterministic parsing: shift-reduce approach, based on the current word and stated, use a classifier to predict the next parsing step -- Malt

Evaluating parsing

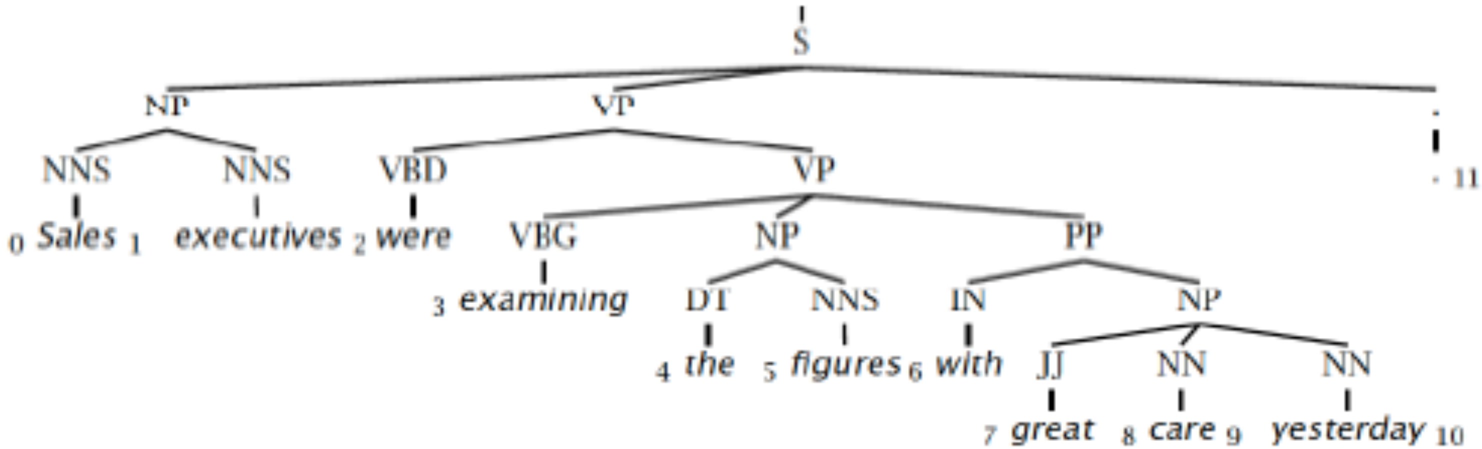


Evaluation of constituency parsing: bracketed P/R/F scores

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)



Evaluation of constituency parsing: bracketed P/R/F scores

Gold 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)

Evaluation of constituency parsing: bracketed P/R/F scores

Gold 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)

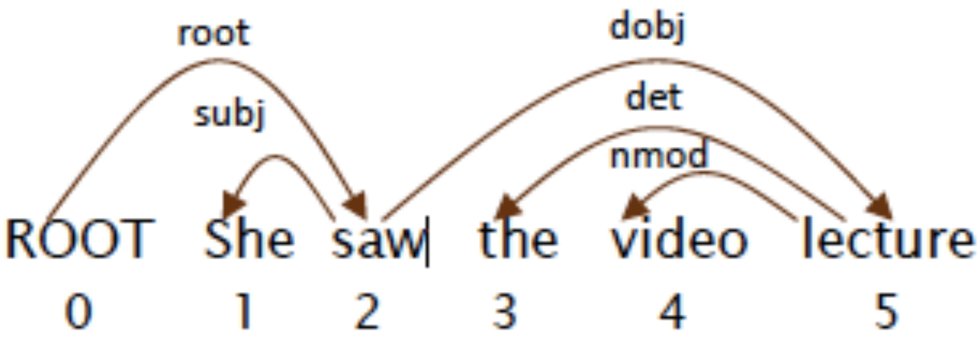
Parseval measures

Labeled Precision: P=3/7=42.9%

Labeled Recall: R=3/8=37.5%

F=40.0%

Evaluation of dependency parsing: labeled dependency accuracy



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

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

Gold

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

Parsed

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

Tools

- Charniak (constituent parser with discriminative reranker)
- Stanford (provides constituent and dependency trees)
- Berkeley (constituent parser with latent variables)
- MST (dependency parser, needs POS tagged input)
- Bohnet's (dependency parser, needs POS tagged input)
- Malt (dependency parser, needs POS tagged input)

Berkeley parser

"Learning Accurate, Compact, and Interpretable Tree Annotation"
Slav Petrov, Leon Barrett, Romain Thibaux and Dan Klein
in COLING-ACL 2006

and

"Improved Inference for Unlexicalized Parsing"
Slav Petrov and Dan Klein
in HLT-NAACL 2007

Downloading

Berkeley parser

http://code.google.com/p/berkeleyparser/

-> parser

-> English grammar

EVALB

http://nlp.cs.nyu.edu/evalb/

-> “make” to install

Sample runs

Running the parser on a toy bnews test set:

```
java  -Xmx2000m -jar
BerkeleyParser-1.7.jar -gr eng_sm6.gr
<prs-lab/data/bn_raw.test >bn_prs.out
```

Running EVALB to assess the performance:

```
./evalb -p sample/sample.prm ../prs-
lab/data/bn_prs.test ../bn_prs.out
```

Does it make sense?

- Evaluation
  - EVALB, in a minute
- Grammar

```
java  -Xmx2000m  -cp
BerkeleyParser-1.7.jar edu/berkeley/
nlp/PCFGLA/WriteGrammarToTextFile
eng_sm6.gr grammartxt
```

Learning a new grammar

```
java  -Xmx2000m  -cp BerkeleyParser-1.7.jar
edu.berkeley.nlp.PCFGLA.GrammarTrainer -path prs-
lab/data/bn_prs.train -out eng_bn.gr -treebank
SINGLEFILE
```

TIPS:

- Don't do it unless needed, precompiled grammars provide a very good performance
- Need a lot of training data!
  WSJ: 1 million tokens, 40k sentences
- Tagsets: data sparsity problem
  You might have to simplify your tagset

Summary

- Constituency vs. Dependency representation
- Grammars, CFG
- Treebanks and Probabilistic CFG
- CKY parsing
- Dependency parsing
- Evaluating parsing
- Parsing tools