8. Syntactic Parsing

Mariana Romanyshyn
Grammarly, Inc.

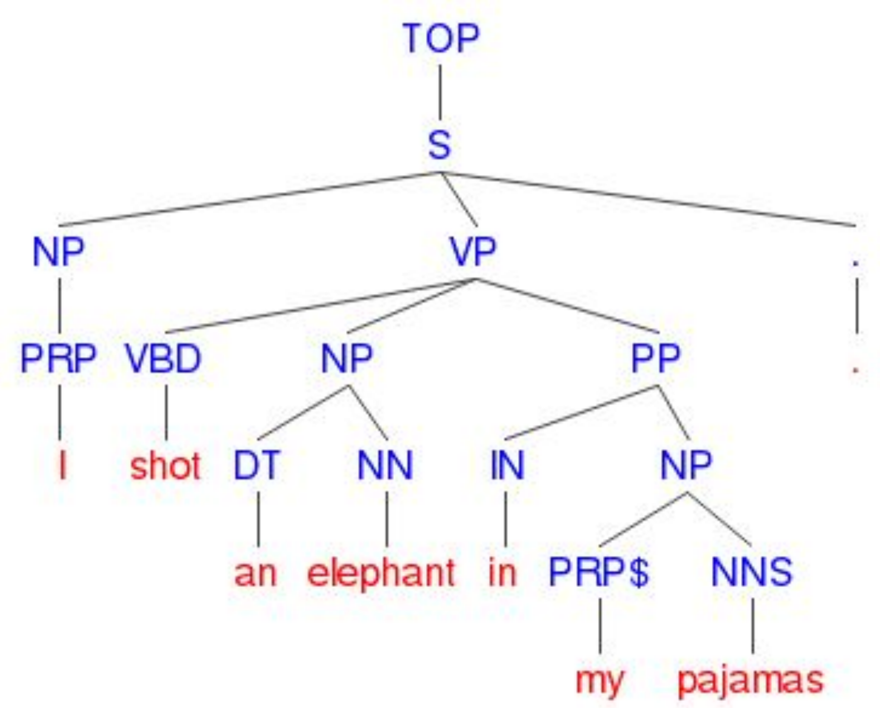
Contents

1. Constituency parsing
   a. algorithms
   b. metrics
2. Dependency parsing
   a. algorithms
   b. metrics
3. Parsing errors

1. Constituency parsing

Constituency parsing

- appeared in 1900s, was formalized in 1950s
- breaks a sentence into independent constituents
- operates at the phrase/clause level
- the tree ends with a ***TOP*** or ***ROOT***



Constituency parsing - bracketed format

```
(TOP (S (PP (IN With)
            (NP (NP (NNS celebrations))
                (PP (IN for)
                    (NP (NP (DT the)
                            (JJ long-anticipated)
                            (NN start))
                        (PP (IN of)
                            (NP (DT the) (NN year) (CD 2000)))))
                (ADVP (RB barely) (RB over))))
        (, ,)
        (NP-TMP (NN today))
        (NP-SBJ-1 (JJ Chinese)
                  (NNS people))
        (VP (VBD began)
            (ADVP (RB busily))
            (VP (VBG preparing)
                (S (NP-SBJ (-NONE- *PRO*-1))
                   (VP (TO to)
                       (VP (VB mark)
                           (NP (DT another) (JJ new)(NN year)))))))
        (. .)))
```

Treebanks

- Benefits:
  - Good for testing linguistic hypotheses
  - Great training data
  - Good evaluation set

- Problems:
  - Costly
  - May contain errors
  - May use different notations

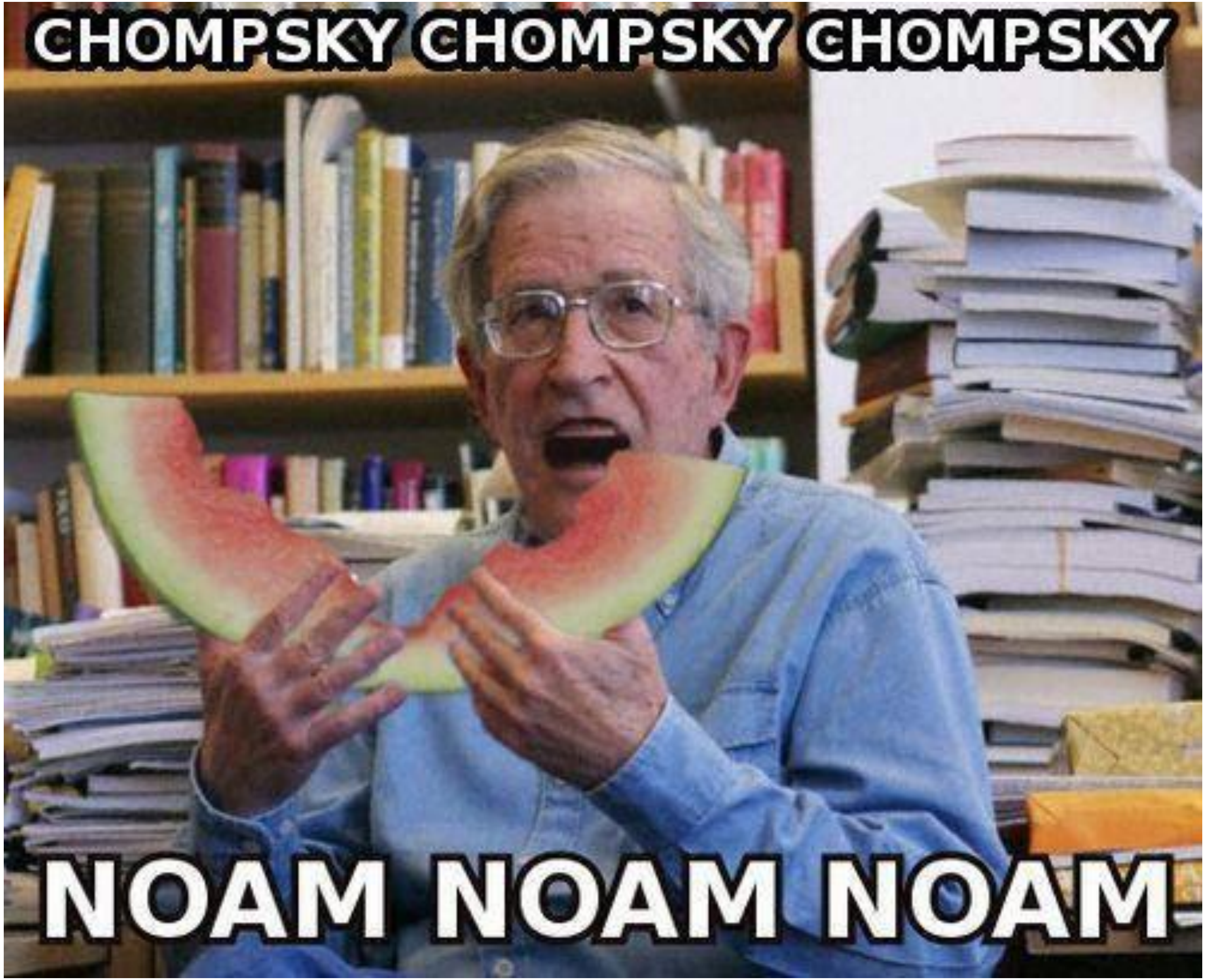
Treebanks

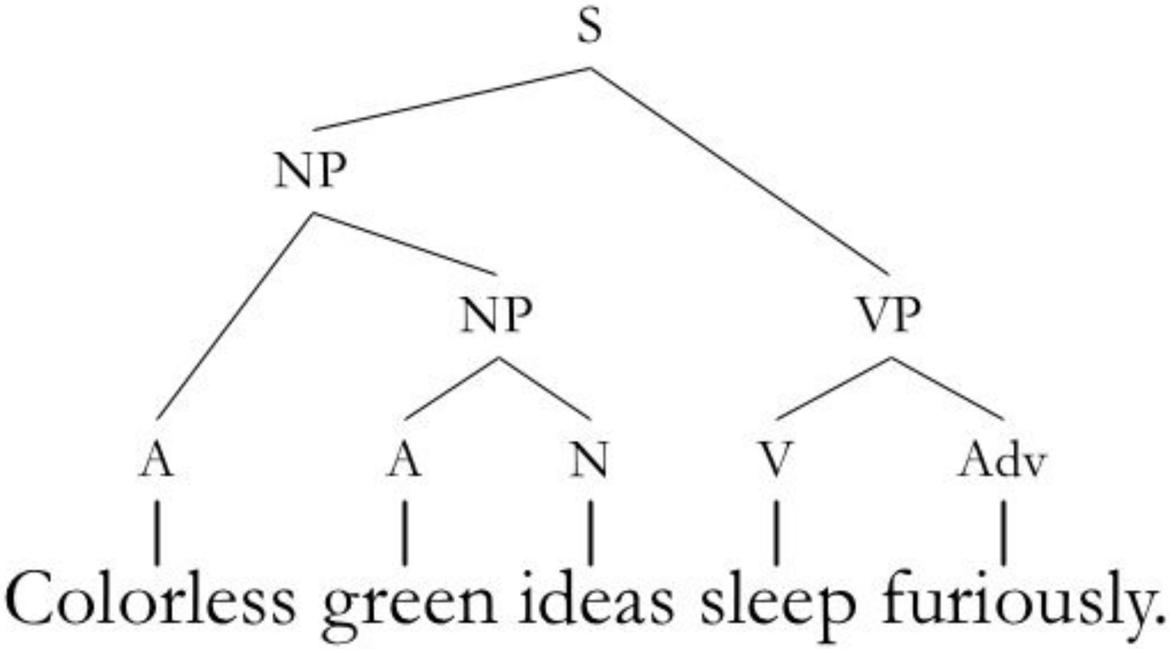
Popular treebanks for the English language:

- Penn Treebank (Brown, Switchboard, ATIS, WSJ)
- Ontonotes 5.0
- English Web Treebank
- QuestionBank
- BNC
- Negra treebank for German

Treebanks

```
(TOP (FRAG (NP (NP (DT The) (JJS best)) (SBAR (WHNP-1 (-NONE- *0*)) (S (NP-SBJ (EX there))
(VP (VBZ is) (NP-PRD-1 (-NONE- *T*)) (PP (IN in) (NP (NN service))))))) (. .)))

(TOP (S (NP-SBJ (PRP I)) (VP (VP (VBD was) (ADVP-TMP (RB recently)) (VP (VBG traveling)
(PP-LOC (IN down) (NP (NNP I-24))) (PP-DIR (IN from) (NP (NNP Nashville))) (PP (IN with)
(NP (PRP$ my) (CD 3) (JJ young) (NNS children))))) (CC and) (VP (VBD had) (NP (DT a) (NN
blowout)) (PP-LOC (IN on) (NP (DT the) (NN southeast) (NN side))))) (. .)))

(TOP (S (S (NP-SBJ (PRP It)) (VP (VBD was) (NP-PRD (CD 4:50)) (SBAR-TMP (WHADVP-9 (WRB
when)) (S (NP-SBJ (DT a) (NN friend)) (VP (VBD told) (NP-1 (PRP me)) (S (NP-SBJ-1 (-NONE-
*PRO*)) (VP (TO to) (VP (VB call) (NP (NNP Bud))))) (ADVP-TMP-9 (-NONE- *T*)))))))) (, ,) (S
(NP-SBJ (PRP he)) (VP (MD would) (VP (VB take) (NP-CLR (NN care)) (PP-CLR (IN of) (NP (PRP
me)))))) (. .)))

(TOP (S (CONJP (RB Not) (RB only)) (SINV (VBD did) (NP-SBJ (PRP they)) (VP (VB answer) (NP
(DT the) (NN phone)) (PP-TMP (IN at) (NP (CD 4:50))) (PP-TMP (IN on) (NP (DT a) (NNP
Thursday))))) (, ,) (S (NP-SBJ-1 (PRP they)) (VP (VBD hit) (NP (DT the) (NN ground)) (S-ADV
(NP-SBJ-1 (-NONE- *PRO*)) (VP (VBG moving))))) (. !)))

...
```


CHOMPSKY CHOMPSKY CHOMPSKY
NOAM NOAM NOAM



vs.

Furiously sleep ideas green colorless.

Context-free grammar

G = (N, ∑, R, S), where

- N – a final set of non-terminal symbols
  {NP, VP, PP, S, SQ, SBAR, …}
- ∑ – a final set of terminal symbols
  {“hi”, “my”, “car”, “kitten”, “decided”, …}
- R – a finite set of rules
  {NP -> NP PP, NP -> NP CC NP,
  VP -> VBZ NP PP, PP -> IN NP, …}
- S – a start symbol for each tree *(TOP/ROOT/S1)*

Context-free grammar

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP
America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP
(VBD increased) (NP (JJ last) (NN year))) (. .)))
```

```
(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved)
(ADVP (RB too))) (. .)))
```

Context-free grammar

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP
America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP
(VBD increased) (NP (JJ last) (NN year))) (. .)))
```

```
(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved)
(ADVP (RB too))) (. .)))
```

N = {S, NP, PP, VP, ADVP}
∑ = {DT, JJ, NN, IN, NNP, CC, NNS, VBD, RB}
S = TOP

Context-free grammar

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP (VBD increased) (NP (JJ last) (NN year))) (. .)))
```

```
(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved) (ADVP (RB too))) (. .)))
```

```
TOP -> S
S -> NP VP .
NP -> NP PP
NP -> NP CC NP
NP -> DT JJ NN
NP -> DT JJ NNS
```

```
NP -> JJ NN
NP -> NNP
VP -> VBD NP
VP -> VBD ADVP
PP -> IN NP
ADVP -> RB
```

Probabilistic context-free grammar

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP
America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP
(VBD increased) (NP (JJ last) (NN year))) (. .)))
```

```
(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved)
(ADVP (RB too))) (. .)))
```

```
TOP -> S            [1]         NP -> JJ NN         [1/7]
S -> NP VP .        [1]         NP -> NNP           [1/7]
NP -> NP PP         [1/7]       VP -> VBD NP        [1/2]
NP -> NP CC NP      [1/7]       VP -> VBD ADVP      [1/2]
NP -> DT JJ NN      [2/7]       PP -> IN NP         [1]
NP -> DT JJ NNS     [1/7]       ADVP -> RB          [1]
```

Issues with PCFGs

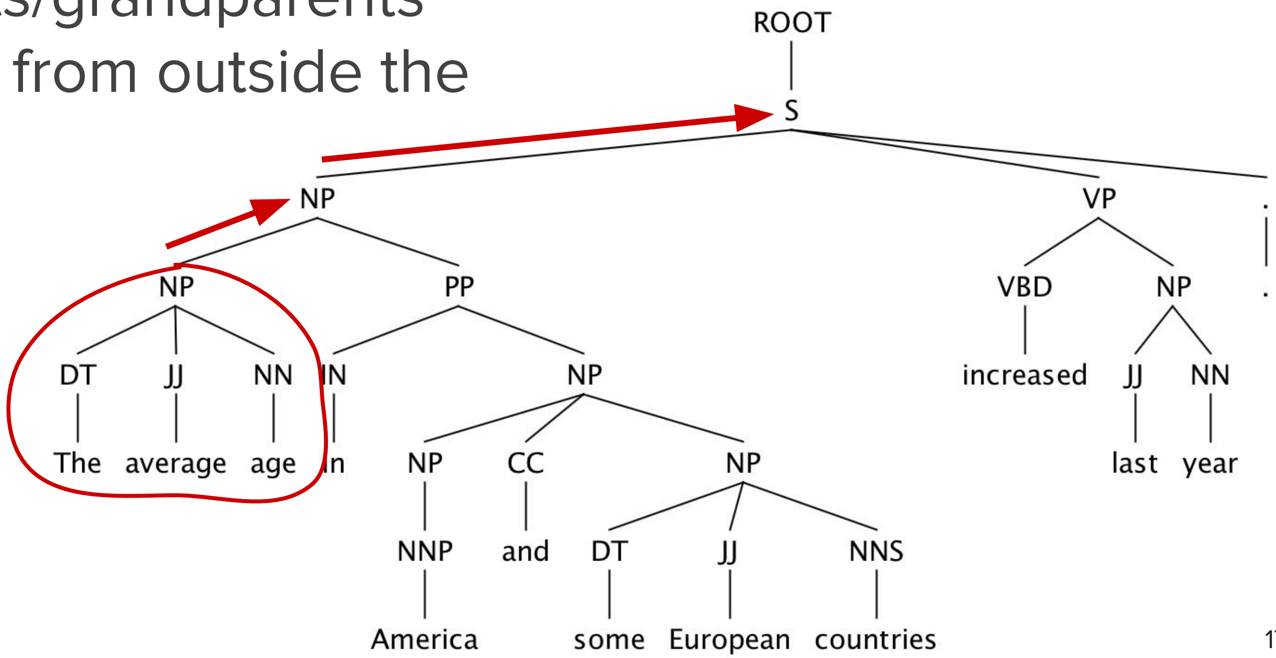
- Poor independence assumptions
  - the probability of the rule is calculated in isolation

- Lack of lexical conditioning
  - don’t model syntactic facts about specific words

Vertical Markovization

Idea:

- encode parents/grandparents
- to add context from outside the phrase



Vertical Markovization

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP
America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP
(VBD increased) (NP (JJ last) (NN year))) (. .)))
```

```
(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved)
(ADVP (RB too))) (. .)))
```

```
TOP    -> S           [1]
S^TOP  -> NP VP .     [1]
NP^S   -> NP PP       [1/2]
NP^PP  -> NP CC NP    [1]
NP^NP  -> DT JJ NN    [1/3]
NP^NP  -> DT JJ NNS   [1/3]
NP^S   -> DT JJ NN    [1/2]
```

```
NP^NP -> NNP          [1/3]
NP^VP -> JJ NN        [1]
VP^S  -> VBD NP       [1/2]
VP^S  -> VBD ADVP     [1/2]
PP^NP -> IN NP        [1]
ADVP^VP -> RB         [1]
```

Vertical Markovization

Pros:

- better disambiguation

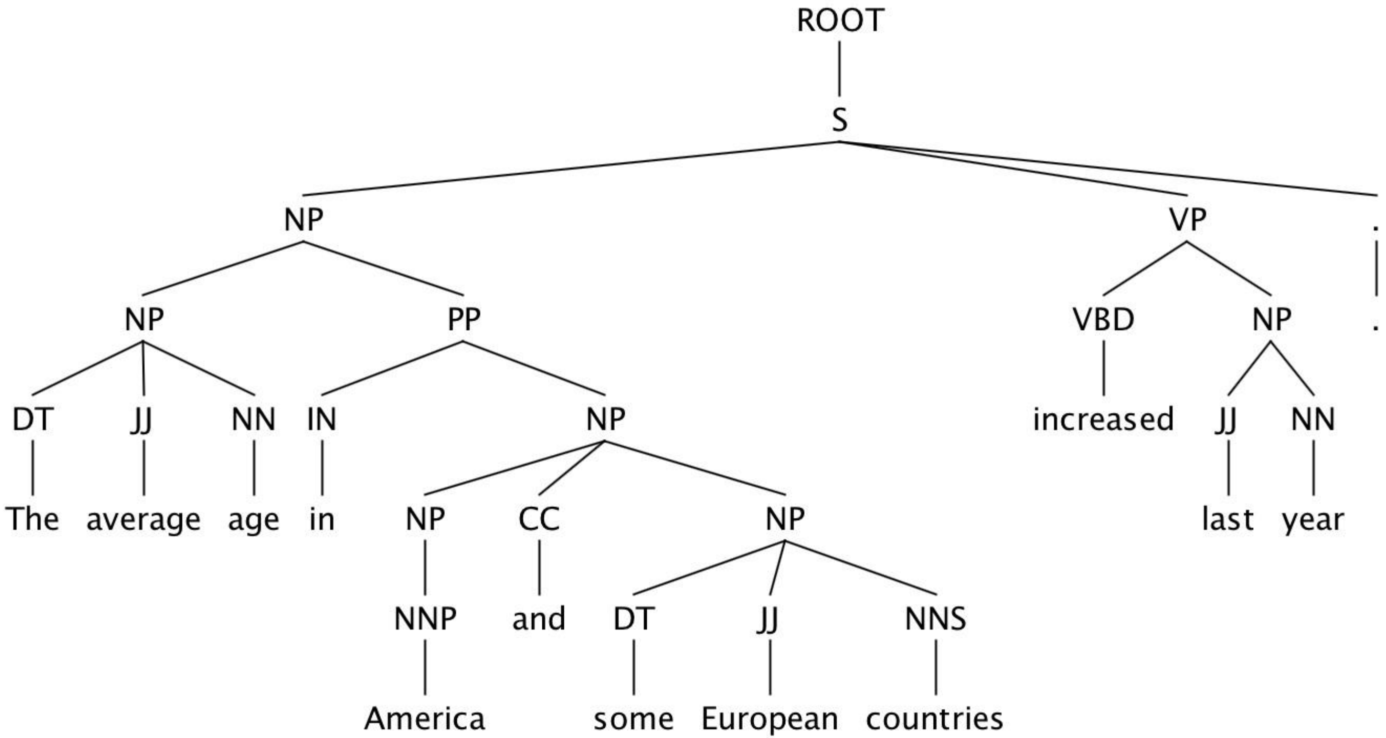
Cons:

- size of the grammar increases
- the amount of training data available for each grammar rule decreases => overfitting

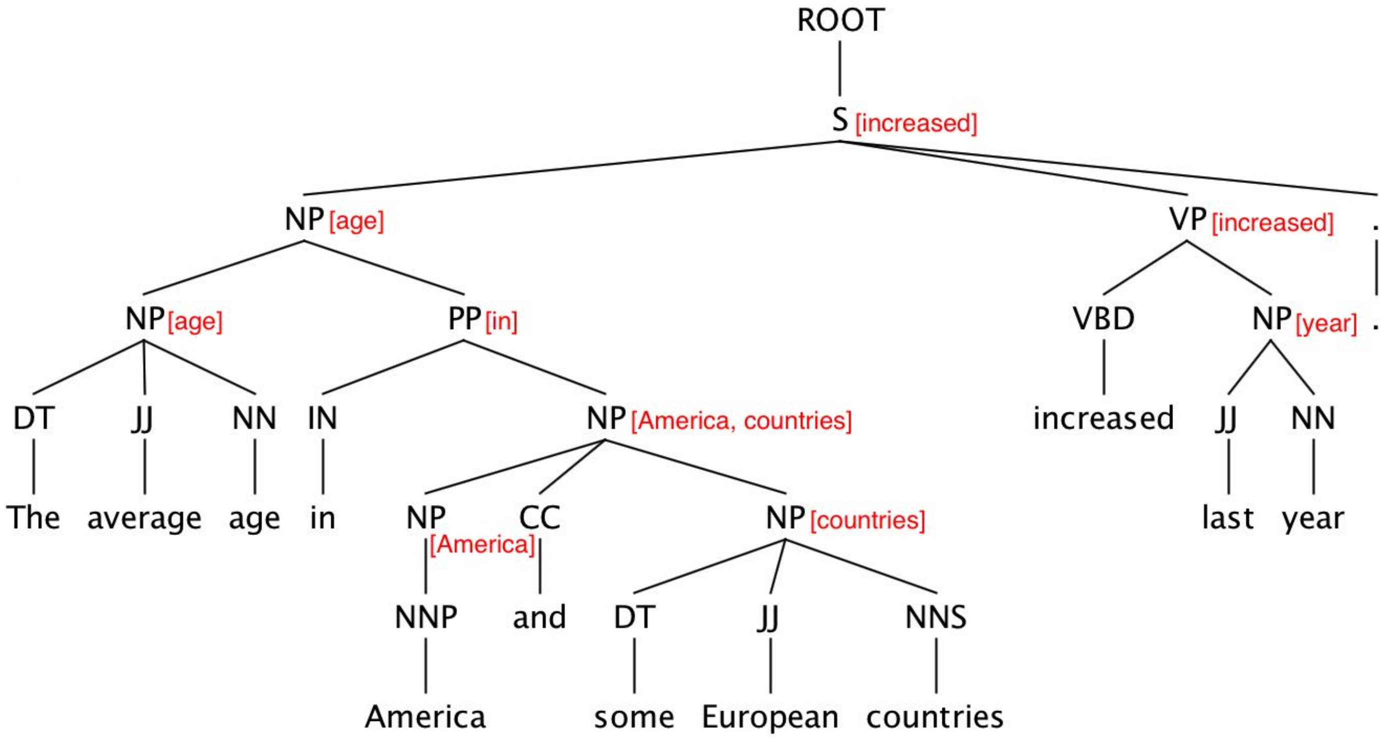
Conclusion:

- find the right level of granularity

Constituency parsing: head nodes



Constituency parsing: head nodes



Constituency parsing: head nodes

For example, let's find the head of NP:

- If the last word is tagged POS, return last-word.
- Else search from right to left for the first child which is an NN, NNP, NNPS, NX, POS, or JJR.
- Else search from left to right for the first child which is an NP.
- Else search from right to left for the first child which is a $, ADJP, or PRN.
- Else search from right to left for the first child which is a CD.
- Else search from right to left for the first child which is a JJ, JJS, RB or QP.
- Else return the last word

Lexicalized PCFG

Lexicalized rules:

```
NP/age -> DT/the JJ/average NN/age
NP/America -> NNP/America
NP/countries -> DT/some JJ/European NNS/countries
NP/age -> NP/age PP/in
NP/year -> JJ/last NN/year
PP/in -> IN/in NP/America+countries
VP/increased -> VBD/increased NP/year
...
```

How to estimate probability? ¯_(ツ)_/¯

Lexicalized PCFG

Not informative at all:

P(`NP/age -> DT/the JJ/average NN/age`) =
    C(`NP/age -> DT/the JJ/average NN/age`) / C(`NP/age`)

A better alternative (Collins parser):

```
P(NP/age->DT/the JJ/average NN/age) = P(head==NN/age|NP/age)
                                     * P(DT/the...|NP/age)
                                     * P(JJ/average...|NP/age)
```

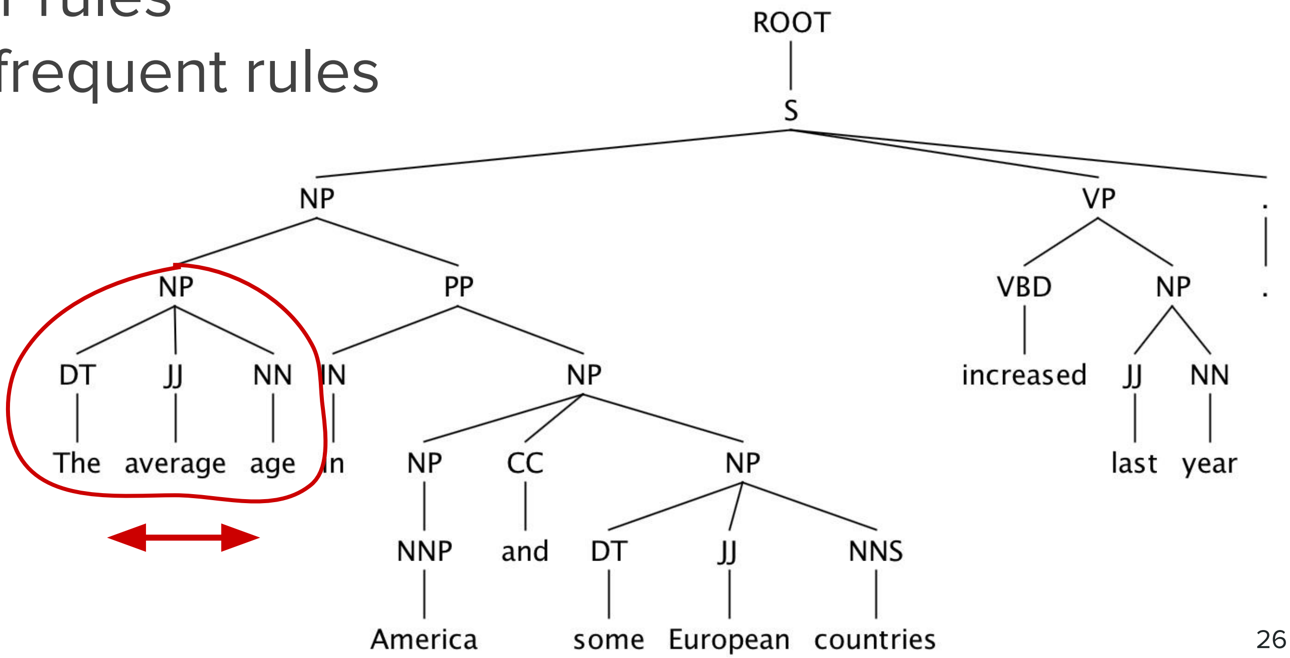
One more tiny problem

NP → DT JJ NN
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
...

Horizontal Markovization

Idea:

- collapse similar rules
- to avoid too infrequent rules



Horizontal Markovization

```
(TOP (S (NP (NP (DT The) (JJ average) (NN age)) (PP (IN in) (NP (NP (NNP America)) (CC and) (NP (DT some) (JJ European) (NNS countries))))) (VP (VBD increased) (NP (JJ last) (NN year))) (. .)))

(TOP (S (NP (DT The) (JJ general) (NN well-being)) (VP (VBD improved) (ADVP (RB too))) (. .)))
```

```
NP -> DT JJ NN    [2/7]   →   NP -> ... NN      [3/7] or
NP -> JJ NN       [1/7]       NP -> ... JJ NN   [3/7]

VP -> VBD NP      [1/2]   →   VP -> VBD ... [1]
VP -> VBD ADVP    [1/2]
```

Constituency parsing algorithms

- Top-down
  - start from ROOT and try to match input sentence

- Bottom-up
  - start from input sentence and try to match ROOT

- Dynamic programming
  - try all combinations and store partial results on the way
  - e.g., CKY, Earley

Top-down constituency parsing: recursive-descent

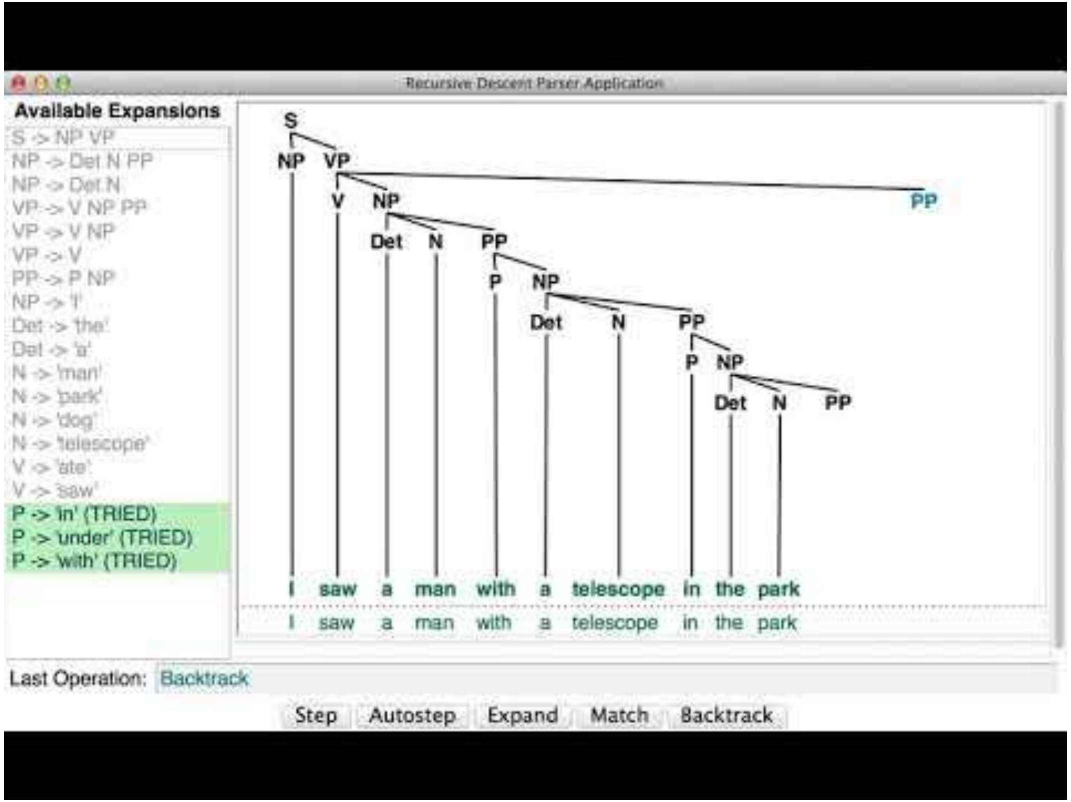
Pros:

- can grasp long-distance relations

Cons

- can go into an endless cycle
- slow due to frequent backoff

Top-down constituency parsing: recursive-descent



Bottom-up constituency parsing: shift-reduce

Data

- ***queue*** - the words of the sentence
- ***stack*** - partially completed trees

Actions

- ***shift*** - move the word from the queue onto the stack
- ***reduce*** - add a new label on top of the first n constituents on the stack

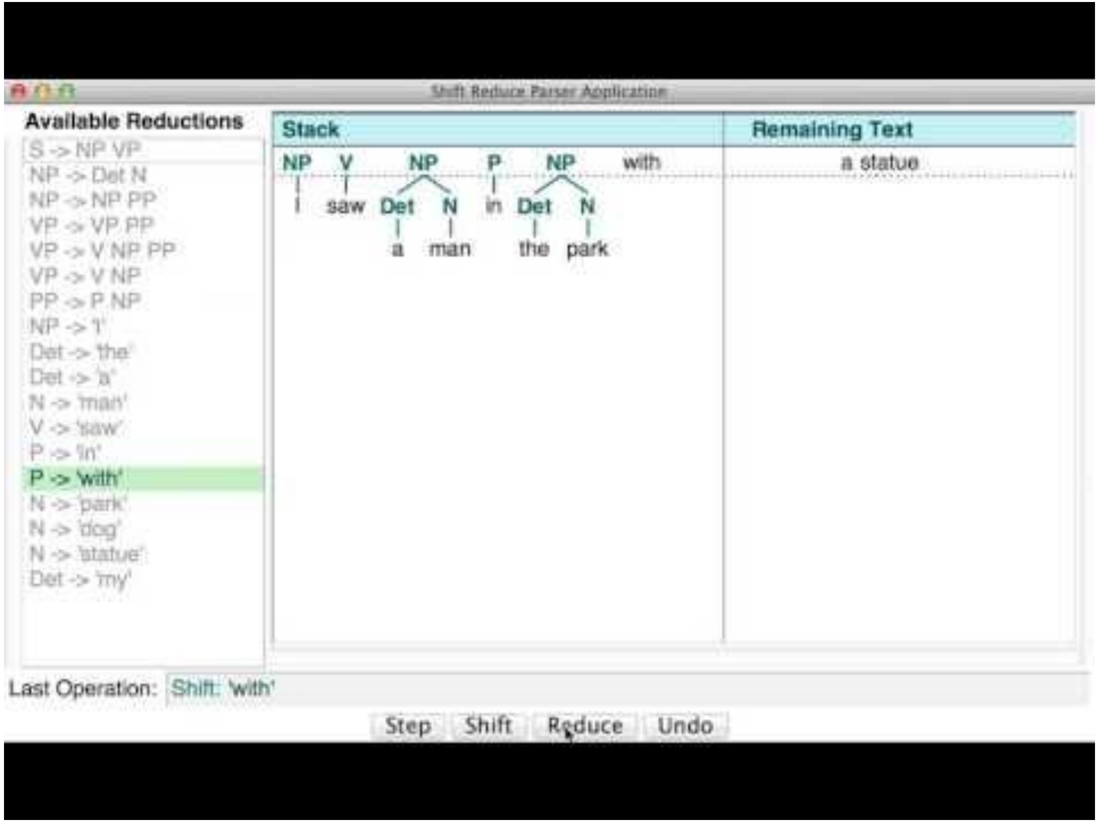
Bottom-up constituency parsing: shift-reduce

Build a parse tree for the sentence below:

A large elephant was wearing my pyjamas

```
S   -> NP VP     [1]
NP  -> DT JJ NN [0.6]
NP  -> PRP$ NN  [0.4]
VP  -> VBD VP   [0.7]
VP  -> VBG NP   [0.3]
```

Bottom-up constituency parsing: shift-reduce demo



Dynamic programming: CKY

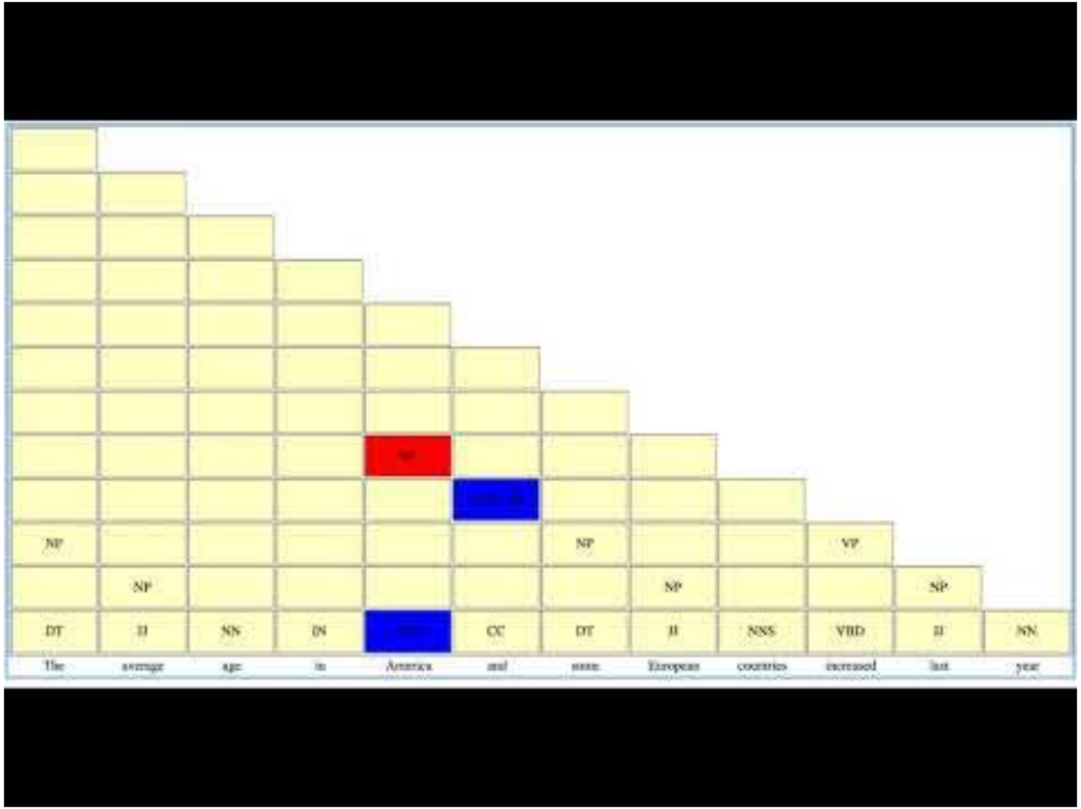
Idea

- build parse tree bottom-up
- combine built trees to form bigger trees using grammar
- find all valid parses with their probabilities

Conditions

- use binary trees only => Chomsky Normal Form
- use dynamic programming

Dynamic programming: CKY



Dynamic programming: CKY

Build the CKY table for the sentence and grammar below:

I saw her duck

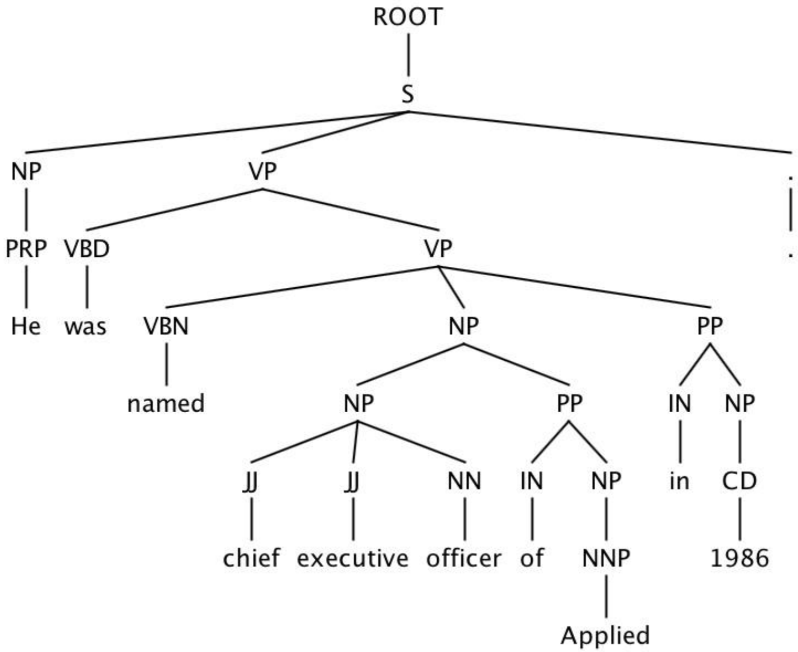
```
S    -> NP VP     [1]            VP   -> “duck”   [0.15]
NP   -> “I”       [0.5]          PRP$ -> “her”    [1]
NP   -> “her”     [0.1]          NN   -> “duck”   [1]
NP   -> PRP$ NN   [0.4]          VBD  -> “saw”    [1]
VP   -> VBD S     [0.05]
VP   -> VBD NP    [0.8]
```

Constituency parsing metrics

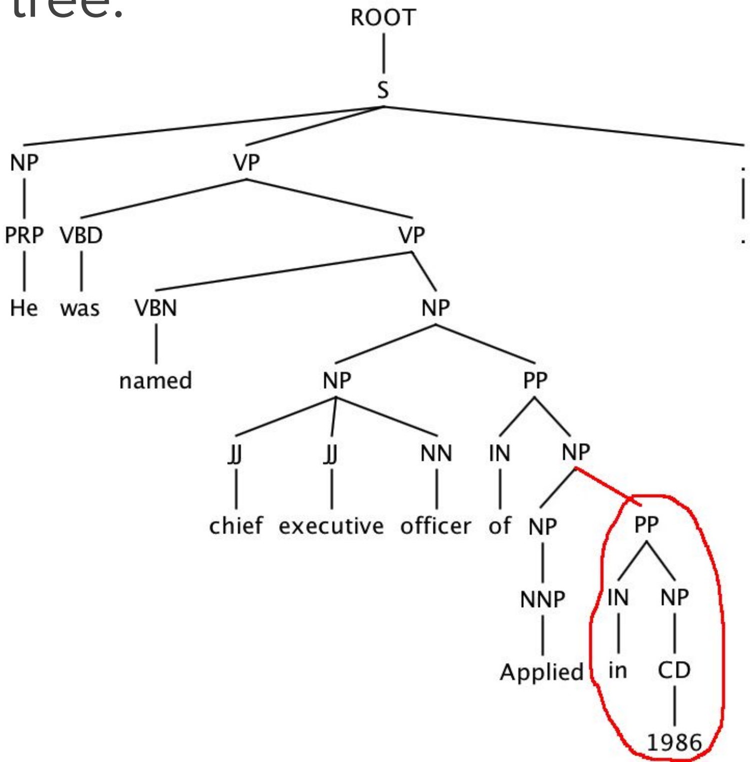
- Parseval
    - percentage of correct nodes (with correct label and span)
- Leaf-Ancestor
    - minimum edit distance of the lineages of the trees
- Minimum Edit Distance
- Cross-Bracketing
    - percentage of brackets that do not coincide in aligned trees
- Recall/Precision/F-measure on separate constituent types
- Complete Match

Constituency parsing metrics

Gold tree:



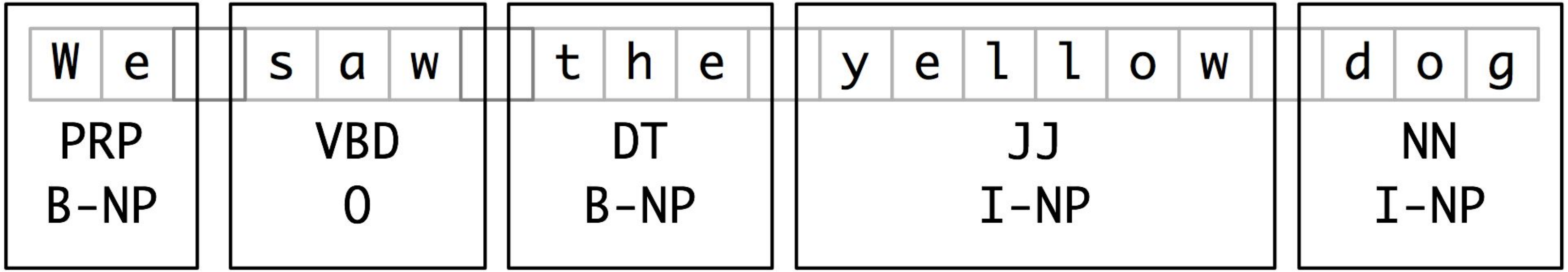
Produced tree:



Chunking

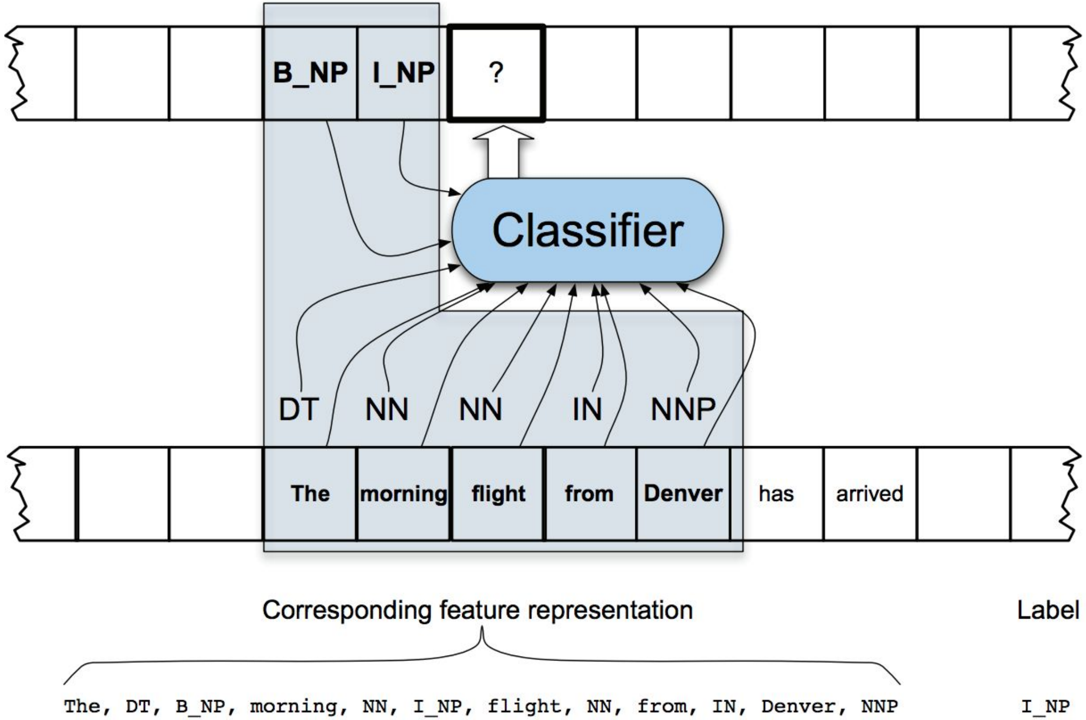
Idea: find and label non-overlapping constituents.

Labels: *NP, VP, PP, ADJP, ADVP.* (BIO-style.)



Example borrowed from http://www.nltk.org/

Chunking

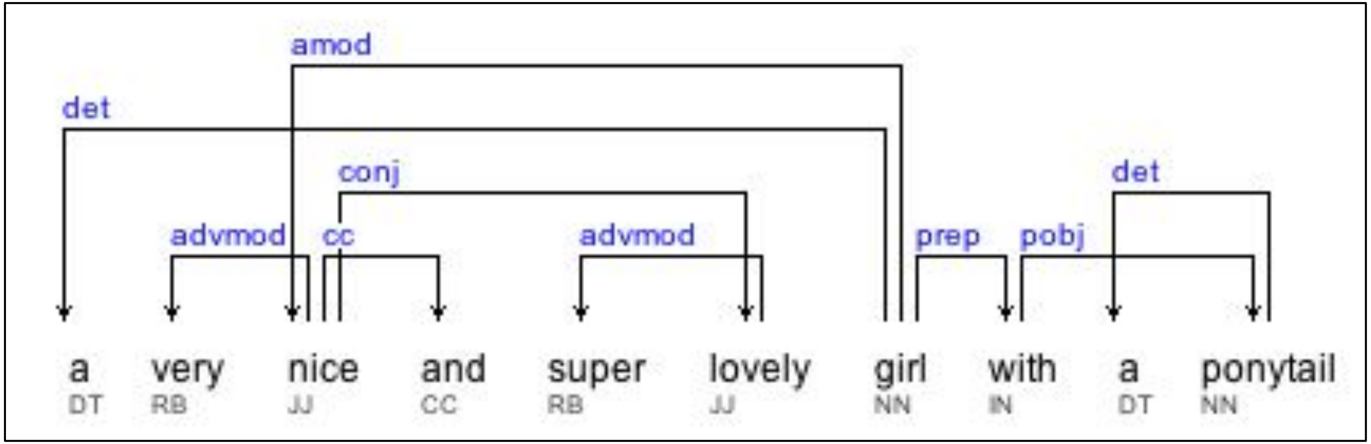


Example borrowed from Jurafsky (2017)

2. Dependency parsing

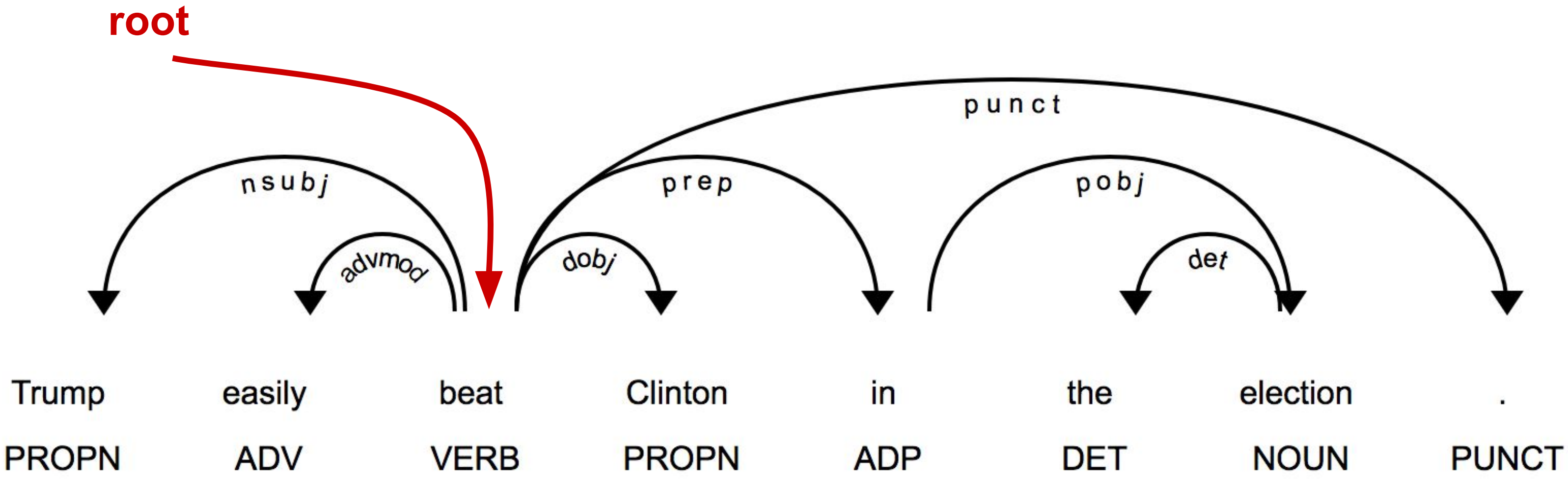
Dependency parsing

- appeared in 2000s
- represents the relations between the words in the sentence
- operates at the word level
- good solution for more synthetic languages



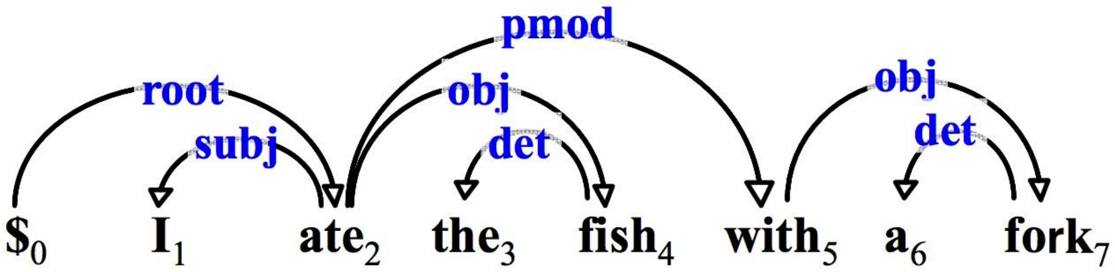
Dependency parsing

- every ***child*** has exactly one ***parent***
- dependencies must form a tree
- the tree ends with ***root***

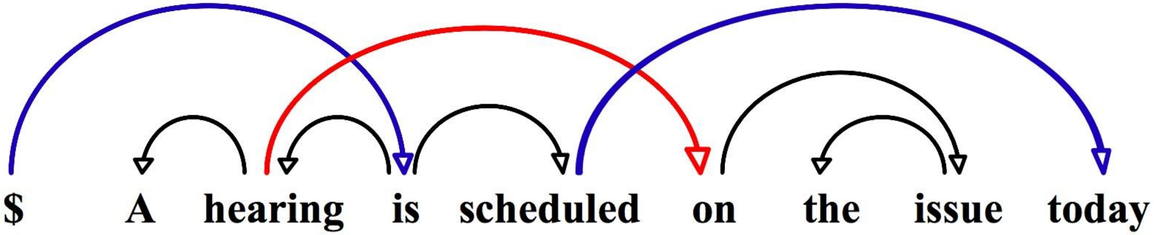


Projectivity

- Projective tree



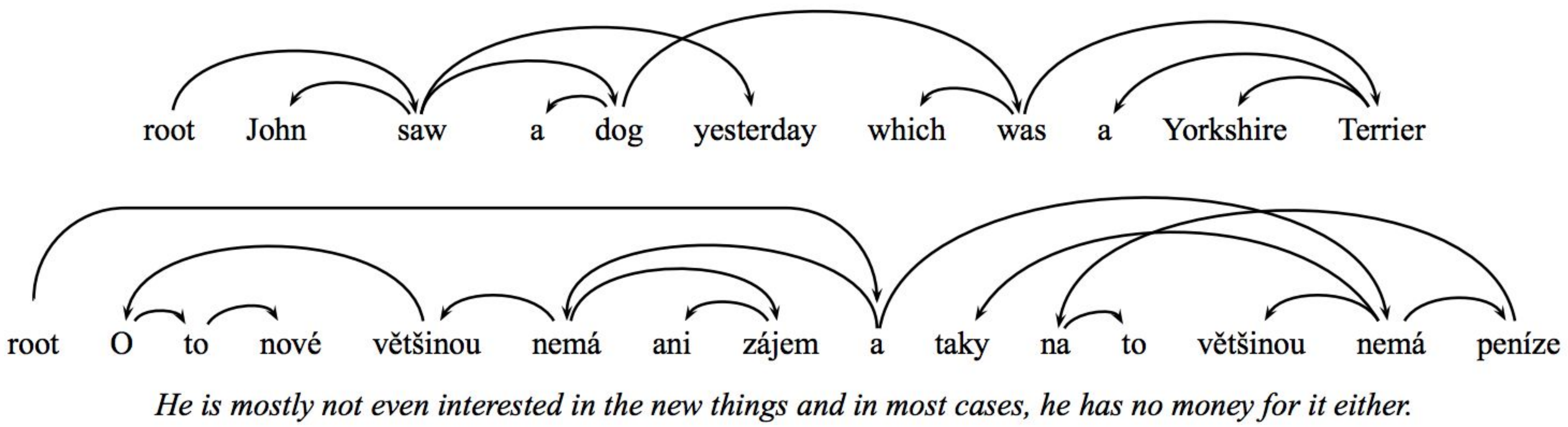
- Non-projective tree



Example from "Dependency Parsing" by Kübler, Nivre, and McDonald, 2009

Projectivity

- Non-projective trees in English and Czech



Borrowed from Ryan McDonald at al. (2005)

Dependency treebanks

- converted from constituency trees using head rules
- Prague Dependency Treebank for Czech
- Universal Dependencies Treebank
  - more than 100 treebanks
  - over 60 languages

Universal Dependency Treebank

1	If	if	IN	3	mark
2	you	you	PRP	3	nsubj
3	want	want	VBP	14	advcl
4	to	to	TO	5	aux
5	receive	receive	VB	3	xcomp
6	e-mails	e-mail	NNS	5	dobj
7	about	about	IN	6	prep
8	my	my	PRP$	10	poss
9	upcoming	upcoming	JJ	10	amod
10	shows	show	NNS	7	pobj
11	,	,	,	14	punct
12	then	then	RB	14	advmod
13	please	please	UH	14	intj
14	give	give	VB	0	root
15	me	me	PRP	14	dative
...					

Graph-based dependency parsing

Idea:

- find the highest score tree from a complete graph.

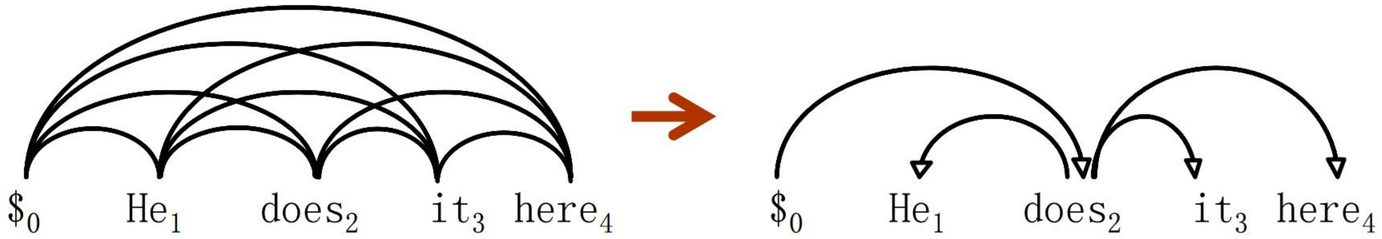
Pros:

- performs better on long-distance dependencies
- allows non-projective trees

Cons:

- slow

Graph-based dependency parsing

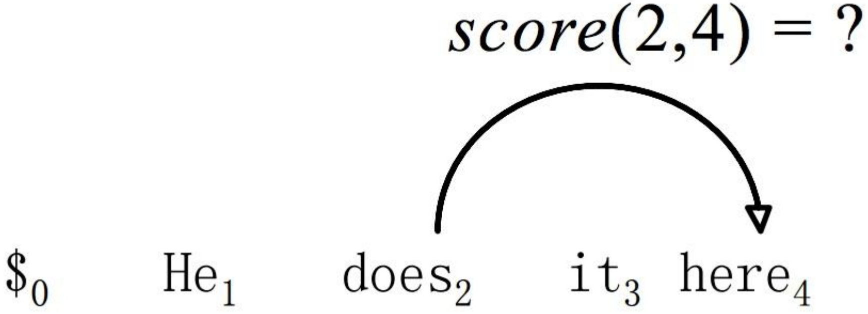


$$Y^* = \arg\max_{Y \in \Phi(X)} score(X, Y)$$

$$score(X, Y) = \sum_{(h,m) \in Y} score(X, h, m)$$

X – sentence
Y – candidate tree
h – head
m – modifier

Features



Each link is a feature vector: ***score(2, 4) = w*f(2,4)***

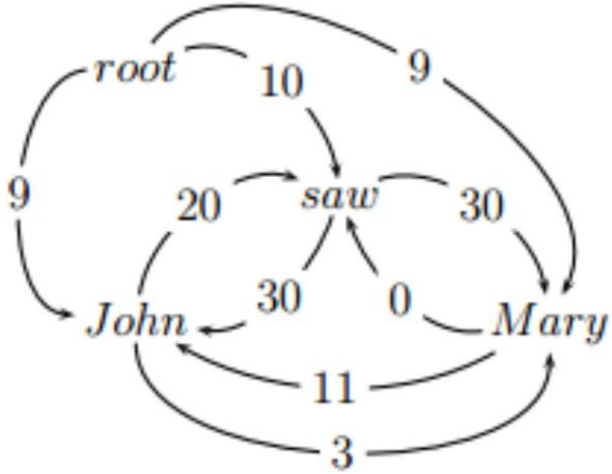
Example from slides of Rush and Petrov (2012)

* As McGwire neared , fans went wild

[went]	[VBD]	[As]	[ADP]	[went]
[VERB]	[As]	[IN]	[went, VBD]	[As, ADP]
[went, As]	[VBD, ADP]	[went, VERB]	[As, IN]	[went, As]
[VERB, IN]	[VBD, As, ADP]	[went, As, ADP]	[went, VBD, ADP]	[went, VBD, As]
[ADJ, *, ADP]	[VBD, *, ADP]	[VBD, ADJ, ADP]	[VBD, ADJ, *]	[NNS, *, ADP]
[NNS, VBD, ADP]	[NNS, VBD, *]	[ADJ, ADP, NNP]	[VBD, ADP, NNP]	[VBD, ADJ, NNP]
[NNS, ADP, NNP]	[NNS, VBD, NNP]	[went, left, 5]	[VBD, left, 5]	[As, left, 5]
[ADP, left, 5]	[VERB, As, IN]	[went, As, IN]	[went, VERB, IN]	[went, VERB, As]
[JJ, *, IN]	[VERB, *, IN]	[VERB, JJ, IN]	[VERB, JJ, *]	[NOUN, *, IN]
[NOUN, VERB, IN]	[NOUN, VERB, *]	[JJ, IN, NOUN]	[VERB, IN, NOUN]	[VERB, JJ, NOUN]
[NOUN, IN, NOUN]	[NOUN, VERB, NOUN]	[went, left, 5]	[VERB, left, 5]	[As, left, 5]
[IN, left, 5]	[went, VBD, As, ADP]	[VBD, ADJ, *, ADP]	[NNS, VBD, *, ADP]	[VBD, ADJ, ADP, NNP]
[NNS, VBD, ADP, NNP]	[went, VBD, left, 5]	[As, ADP, left, 5]	[went, As, left, 5]	[VBD, ADP, left, 5]
[went, VERB, As, IN]	[VERB, JJ, *, IN]	[NOUN, VERB, *, IN]	[VERB, JJ, IN, NOUN]	[NOUN, VERB, IN, NOUN]
[went, VERB, left, 5]	[As, IN, left, 5]	[went, As, left, 5]	[VERB, IN, left, 5]	[VBD, As, ADP, left, 5]
[went, As, ADP, left, 5]	[went, VBD, ADP, left, 5]	[went, VBD, As, left, 5]	[ADJ, *, ADP, left, 5]	[VBD, *, ADP, left, 5]
[VBD, ADJ, ADP, left, 5]	[VBD, ADJ, *, left, 5]	[NNS, *, ADP, left, 5]	[NNS, VBD, ADP, left, 5]	[NNS, VBD, *, left, 5]
[ADJ, ADP, NNP, left, 5]	[VBD, ADP, NNP, left, 5]	[VBD, ADJ, NNP, left, 5]	[NNS, ADP, NNP, left, 5]	[NNS, VBD, NNP, left, 5]
[VERB, As, IN, left, 5]	[went, As, IN, left, 5]	[went, VERB, IN, left, 5]	[went, VERB, As, left, 5]	[JJ, *, IN, left, 5]
[VERB, *, IN, left, 5]	[VERB, JJ, IN, left, 5]	[VERB, JJ, *, left, 5]	[NOUN, *, IN, left, 5]	[NOUN, VERB, IN, left, 5]

Graph-based dependency parsing

- Speed:
  - dynamic programming
  - maximum directed spanning tree



Transition-based dependency parsing

Idea:

- apply transition actions one by one from left to right

Pros:

- fast

Cons:

- performs worse on long-distance dependencies
- allows only projective trees

Transition-based parsing

Configurations:

- ***queue*** - the words of the sentence
- ***stack*** - words yet without head
- ***set of relations***

Transition-based parsing (Arc-Eager)

Actions:

- ***shift*** - move the word from the queue onto the stack
- ***right-arc*** - create a right dependency arc between the word on top of the stack and the next token in the queue
- ***left-arc*** - create a left dependency arc between the word on top of the stack and the next token in the queue
- ***reduce*** - pop the stack, removing only its top item, as long as that item has a head

Transition-based parsing

$$Y^* = \arg\max_{Y \in \Phi(X)} score(X, Y)$$

$$= \arg\max_{a_0 \dots a_m \to Y} \sum_{i=0}^{m} score(X, h_i, a_i)$$

X – sentence

Y – candidate tree

a – action

h – partial result built so far

m – number of words in the sentence (== number of actions)

Transition-based parsing

Now:

- do a sequence of actions through the space of possible configurations
- apply an actions to a configuration and produce a new configuration

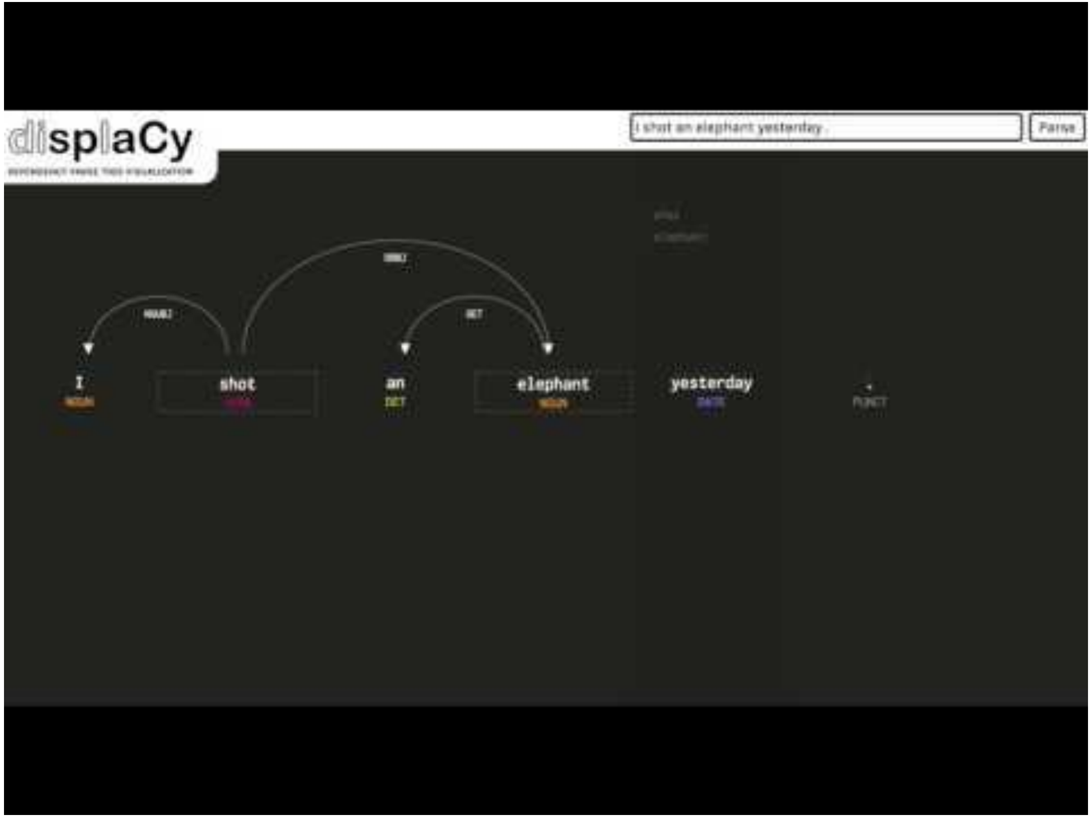
```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)         ; choose a transition operator to apply
    state ← APPLY(t, state)   ; apply it, creating a new state
  return state
```

Transition-based parsing

Build a parse tree for the sentence below:

A large elephant was wearing my pyjamas

Transition-based parsing: demo



Transition-based parsing (Arc-Eager)

Actions:

- ***shift*** - move the word from the queue onto the stack
- ***right-arc*** - create a right dependency arc between the word on top of the stack and the next token in the queue
- ***left-arc*** - create a left dependency arc between the word on top of the stack and the next token in the queue
- ***reduce*** - pop the stack, removing only its top item, as long as that item has a head
- ***swap*** - exchange the words on top of the stack and on top of the queue

Training a transition-based parser

```
training_set ← []
for sentence,tree pair in corpus do
    sequence ← oracle(sentence, tree)
    configuration ← initialize(sentence)
    while not configuration.IsFinal() do
        action ← sequence.next()
        features ← φ(configuration)
        training_set.add(features, action)
        configuration ← configuration.apply(action)
train a classifier on training_set
```

Oracles

Oracle - a function that retrieves the transition at each point in tree.

- static oracle
  - checks: left/right arc => reduce => shift
  - returns the first satisfactory transition
- non-deterministic oracle
  - checks: left/right arc, reduce, shift
  - returns all ***valid*** transitions

Oracles

Oracle - a function that retrieves the transition at each point in tree.

- dynamic
  - train a classifier to decide on the action
  - use golden tree for training
  - return transactions with the lowest loss

Dependency parsing metrics

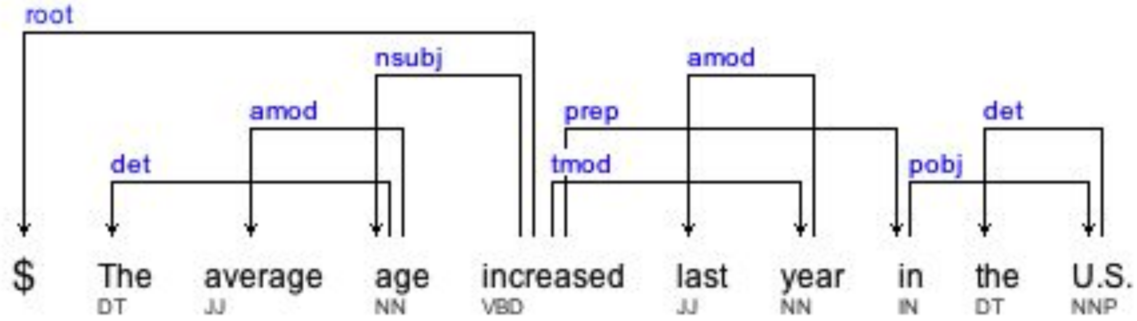
- Unlabeled Attachment Score
    - percentage of words that have correct heads

- Labeled Attachment Score
    - percentage of words that have correct heads and labels

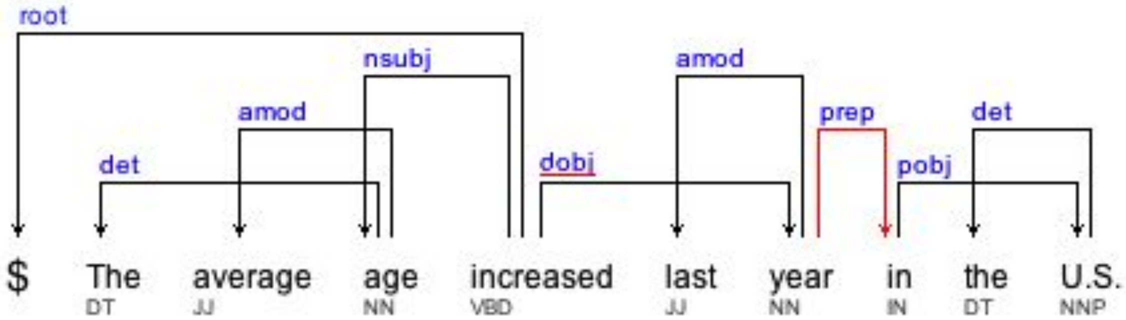
- Recall/Precision/F-measure on separate labels
- Root Accuracy
- Complete Match

Dependency parsing metrics

- Gold tree

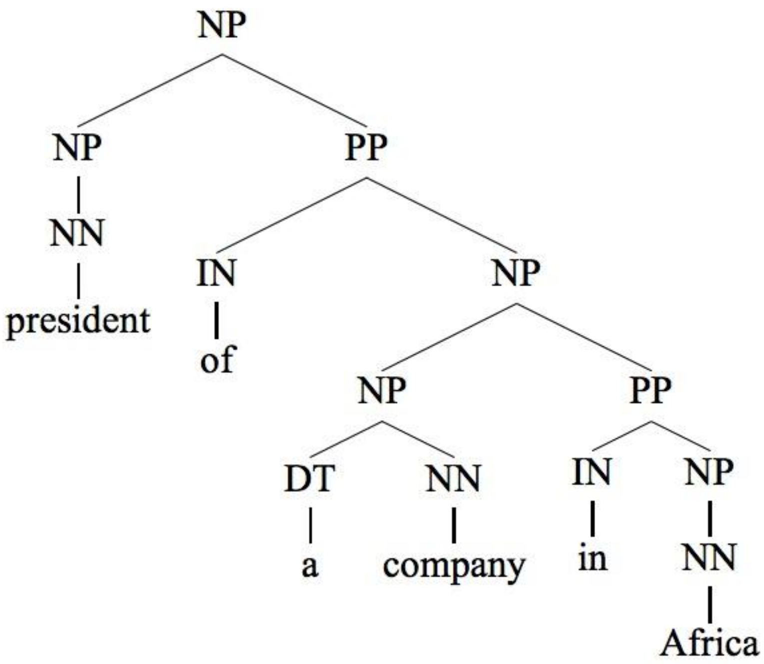


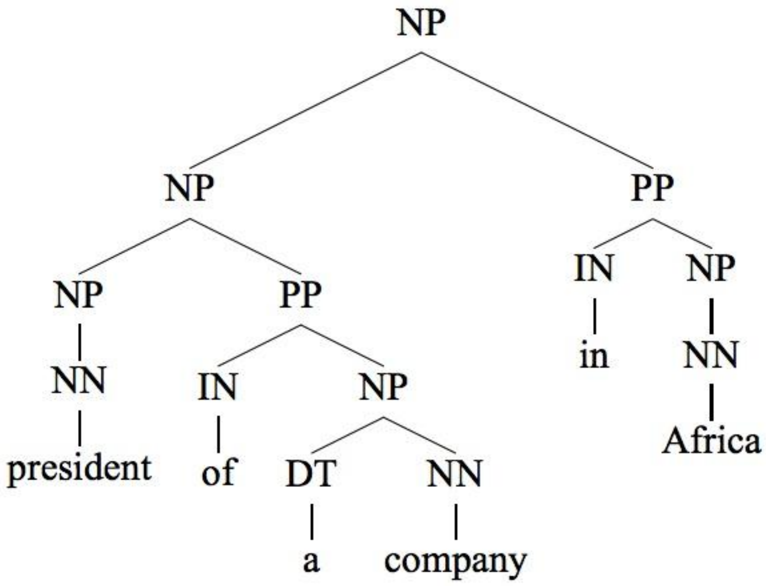
- Produced tree



3. Parsing errors

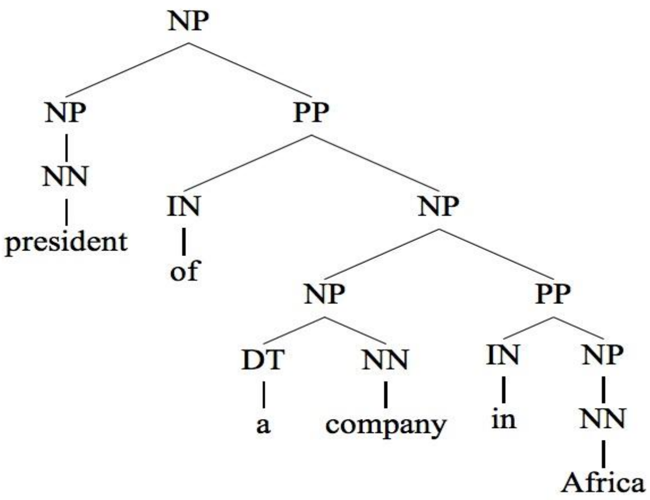
The attachment problem

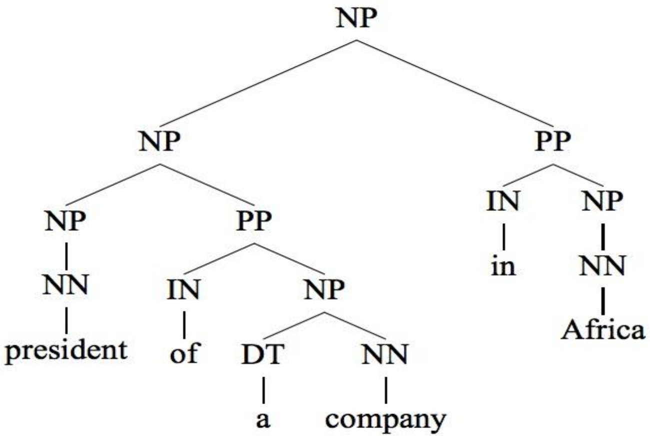




The attachment problem

- PP attachment





The attachment problem

- PP attachment
- NP attachment
  - *We [decided to [build a museum this week]].*
  - *We [decided to [build a museum] this week].*

The attachment problem

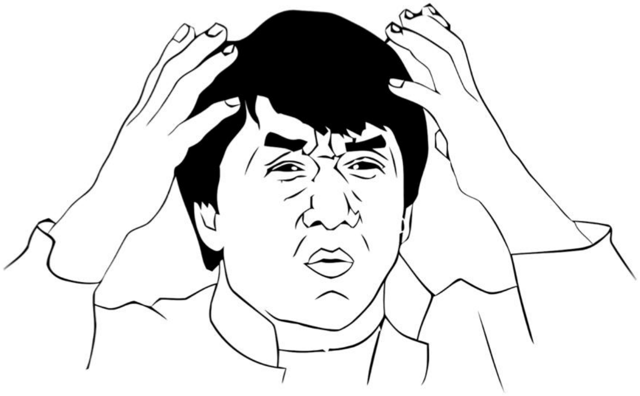
- PP attachment
- NP attachment
- Modifier attachment
    - *[[beautiful women] and men]*
    - *[old [women and men]]*

The attachment problem

- PP attachment
- NP attachment
- Modifier attachment
- Clause attachment
    - *[[I’m glad I’m a man], and [so is Lola]].*
    - *[I’m glad [[I’m a man], and [so is Lola]]].*

The attachment problem

- PP attachment
- NP attachment
- Modifier attachment
- Clause attachment
    - *Іхтіандр врятував дівчину від акули, з якою потім познайомився.*



The attachment problem

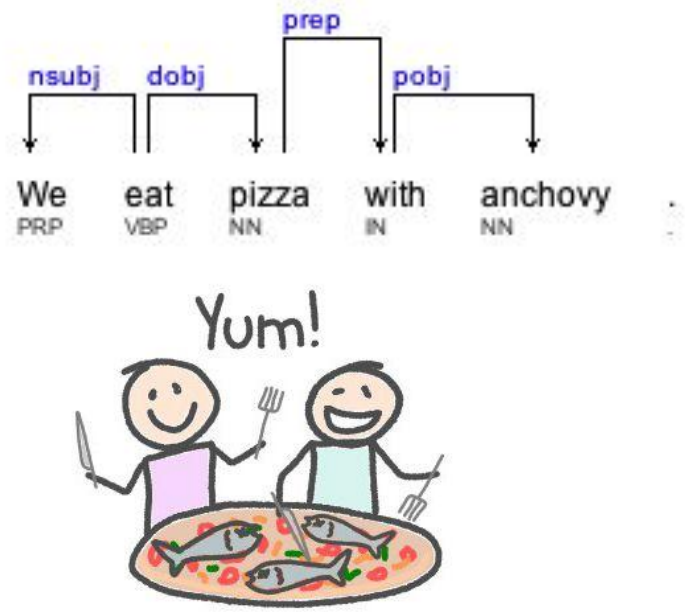
- PP attachment
- NP attachment
- Modifier attachment
- Clause attachment
- VP attachment *(esp. in catenative coordinate structures)*
  - *We have [to pay Tom [[to do the job] and [to manage everything]]].*
  - *We have [[to pay Tom [to do the job]] and [to manage everything]].*

The attachment problem

- PP attachment
- NP attachment
- Modifier attachment
- Clause attachment
- VP attachment *(esp. in catenative coordinate structures)*

```
От було взяло заманулося піти спробувати навчитися
готувати їсти. 8 дієслів підряд. #ГраничнаМова
```

The attachment problem



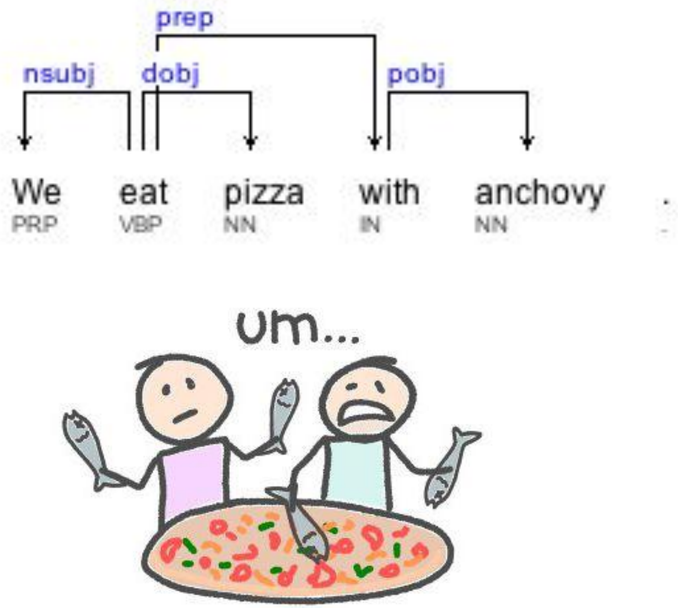


Image by James Constable

The PP attachment problem: solutions

- Majority class (noun attachment) wins
- Most likely class for each preposition wins
- Binary classification using maximum likelihood estimation

1. **If**[3] $f(v, n1, p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, p, n2)}{f(v, n1, p, n2)}$$

2. **Else if** $f(v, n1, p) + f(v, p, n2) + f(n1, p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, p) + f(1, v, p, n2) + f(1, n1, p, n2)}{f(v, n1, p) + f(v, p, n2) + f(n1, p, n2)}$$

3. **Else if** $f(v, p) + f(n1, p) + f(p, n2) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, p) + f(1, n1, p) + f(1, p, n2)}{f(v, p) + f(n1, p) + f(p, n2)}$$

4. **Else if** $f(p) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, p)}{f(p)}$$

5. **Else** $\hat{p}(1|v, n1, p, n2) = 1.0$ (default is noun attachment).

Taken from Collins and Brooks (1995)

The PP attachment problem: solutions

- Majority class (noun attachment) wins
- Most likely class for each preposition wins
- Binary classification using maximum likelihood estimation:
    - *P(eat, pizza, with, anchovy)*
    - *P(eat, pizza, with), P(eat, with, anchovy), P(pizza, with, anchovy)*
    - *P(eat, with), P(with, anchovy), P(pizza, with)*
    - *P(with)*

The coordination attachment problem: solutions

- The closer relation wins
- Similarity of head nodes in coordination
  - *books about musical instruments and other literature*
  - *dogs in houses and cats*
  - *cats with fleas and dogs*
  - *men who like shopping and women*

More things to improve

- Fixing POS errors while building trees
- Exploring richer features
  - *e.g., mark coordination, grandparents, siblings*
- Reranking of n-best parse trees
  - *lexicalization, ancestors, functional/lexical heads*
  - *tree ngrams, rightmost-branch bias*
  - *coordination parallelism*
- Ensembles of parsers
- Semi-supervised learning
- Beam search

References

- Speech and Language Processing, Chapters 11-14, Jurafsky and Martin (2017)
- Syntax and Parsing, Yoav Goldberg, (2017)
- Prepositional Phrase Attachment through a Backed-Off Model, Michael Collins and James Brooks (1995)
- Accurate Unlexicalized Parsing, Dan Klein and Chris Manning (2003)
- Non-projective Dependency Parsing using Spanning Tree Algorithms, Ryan McDonald et al. (2005)
- Improvements in Transition Based Systems for Dependency Parsing, Francesco Sartorio (2015)
- Parsing English in 500 Lines of Python, Matthew Honnibal (2013)
- The Dirty Little Secret of Constituency Parser Evaluation, Romanyshyn and Dyomkin (2014)

Thank you !

Any questions ?