Natural Language Parsing

Simone Filice
filice.simone@gmail.com

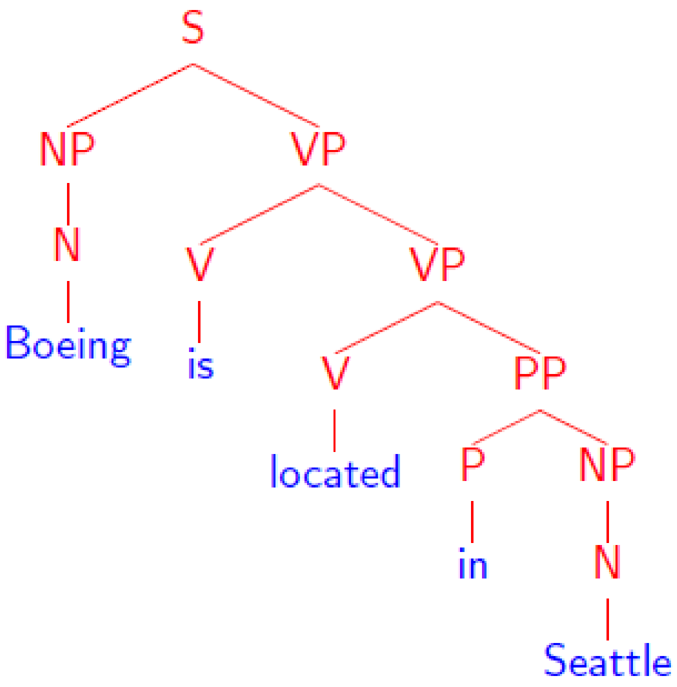
University of Roma Tor Vergata

Material extracted from lectures of Collins and Nivre

Web Mining e Retrieval 2013/2014

Problem Definition

- PARSING: Breaking down a text into its component parts of speech (according to a formal grammar) with an explanation of the form, function, and syntactic relationship of each part
- INPUT: *Boeing is located in Seattle*
- OUTPUT:



CONSTITUENT PARSE TREE

An Example Applications

- In Machine Translation each language has its own word ordering rules
  - English word order is: subject-verb-object
  - Japanes word order is subject-object-verb

- Examples:
  - English: IBM bought Lotus
  - Japanese: *IBM Lotus bought*

  - English: Sources said that IBM bought Lotus yesterday
  - Japanese: *Sources yesterday IBM Lotus bought that said*

Overview

- Context Free Grammars
  - Ambiguity Problem
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Overview

- **Context Free Grammars**
  - Ambiguity Problem
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Context Free Grammars (CFG)

- Formal Definition: a context free grammar (CFG) is a 4-tuple G=(*N*, *Σ*, *R*, *S*) where:
  - *N* is a set of non-terminal symbols
  - *Σ* is a set of terminal symbols
  - *R* is a set of rules of the form $X \rightarrow Y_1 Y_2 \cdots Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
  - $S \in N$ is a distinguished start symbol

A Simple CFG for English

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

$R =$

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

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

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

Left-Most Derivations

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

Left-Most Derivations

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

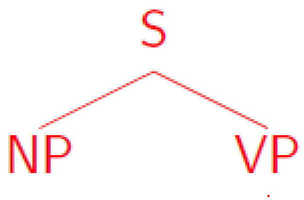
- Example: [S]

S

Left-Most Derivations

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

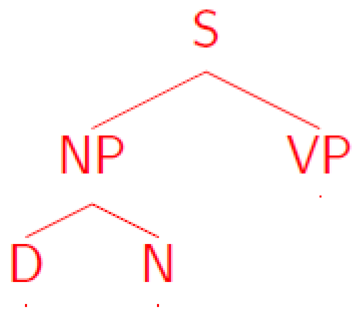
- Example: [S],[NP VP]



Left-Most Derivations

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

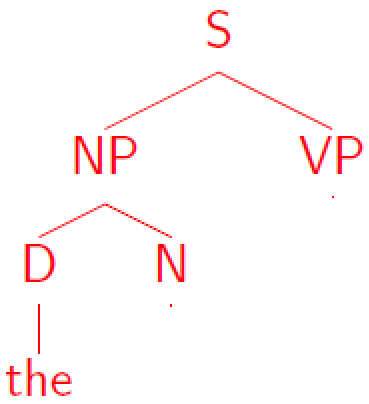
- Example: [S],[NP VP], [D N VP]



Left-Most Derivations

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

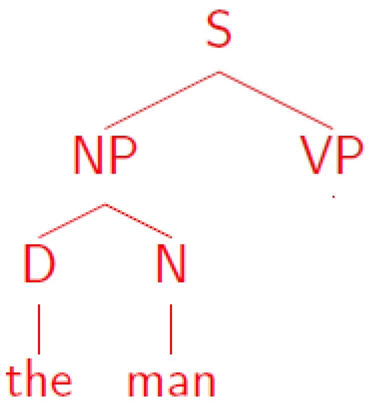
- Example: [S],[NP VP], [D N VP],[the N VP]



Left-Most Derivations

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

- Example: [S],[NP VP], [D N VP],[the N VP], [the man VP]

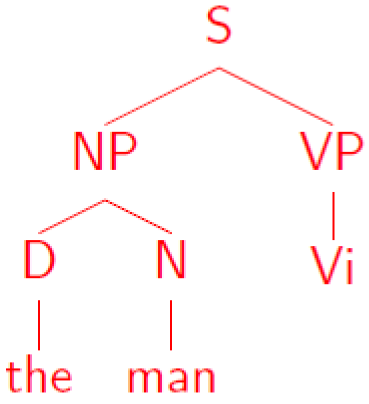


Left-Most Derivations

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

- Example: [S],[NP VP], [D N VP],[the N VP],

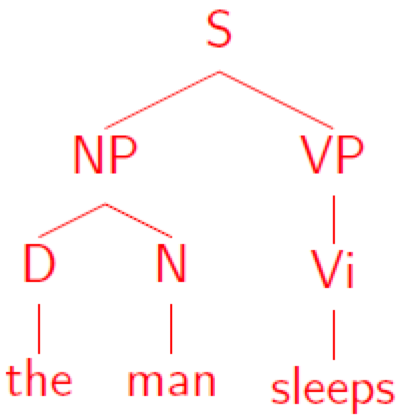
  [the man VP],[the man Vi]



Left-Most Derivations

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

- Example: [S],[NP VP], [D N VP],[the N VP],
  [the man VP],[the man Vi],
  [the man sleeps]



Properties of a CFG

- A Context-free Grammar G defines a set of derivations
- A word $s \in \Sigma^*$ is in the language defined by G if there is at least one derivation that yields *s*
- Each string in the language generated by the CFG may have more than one derivation (*ambiguity problem*)

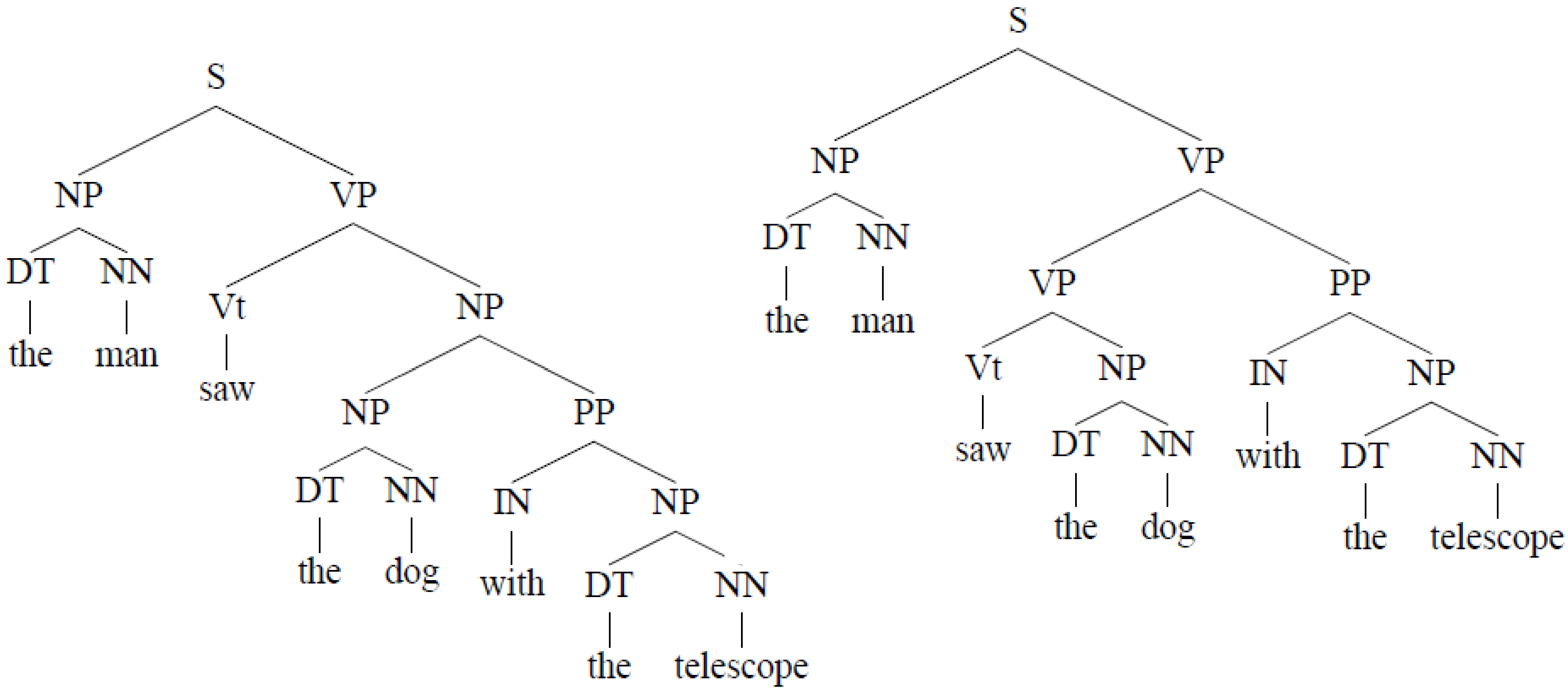
Overview

- **Context Free Grammars**
  - **Ambiguity Problem**
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Ambiguity Problem

INPUT: The man saw the dog with the telescope

POSSIBLE OUTPUTS:

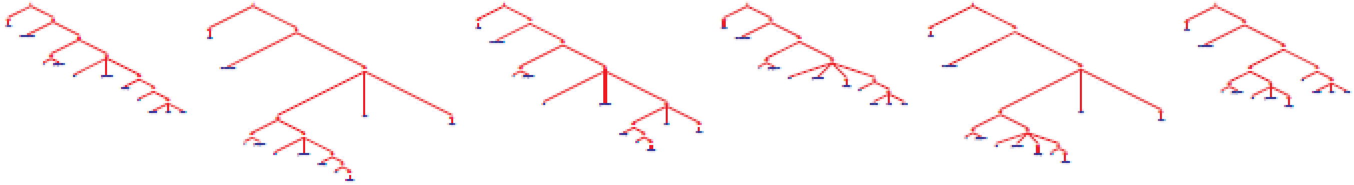


Ambiguity Problem

INPUT:
She announced a program to promote safety in trucks and vans

⇓

POSSIBLE OUTPUTS:



And there are more...

Overview

- Context Free Grammars
  - Ambiguity Problem
- **Probabilistic Context Free Grammars**
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Solving Ambiguity Problem

- Given a sentence *s*, and a formal grammar G, there can be many derivations that yield *s*

- Let $\mathcal{T}_G(s)$ be the set of possible derivations that yield s

- Defining a probability distribuition p(t) over all the possible derivations t ∈ $\mathcal{T}_G(s)$ we are able to disambiguate the parsing problem selecting the most probable parse tree:

$$t^* = \operatorname*{argmax}_{t \in \mathcal{T}_G(s)} p(t)$$

Probabilistic Context-Free Grammars (PCFG)

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

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

- Probability of a tree t with rules

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

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

Deriving PCFG From a Corpus

- Given a set of example trees (a treebank), the underlying CFG can simply be all rules seen in the corpus

- Maximum-likelihood estimation of the probability parameters $q(\alpha \rightarrow \beta)$:

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

  where the counts are taken from a training set of example trees

Overview

- Context Free Grammars
  - Ambiguity Problem
- **Probabilistic Context Free Grammars**
  - **CYK parsing algorithm**
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Chomsky Normal Form

- a Context-Free Grammar G=(*N*, *Σ*, *R*, *S*) in Chomsky Normal Form is as follow:
  - *N* is a set of non-terminal symbols
  - *Σ* is a set of terminal symbols
  - *R* is a set of rules which take one of two forms:
    - $X \rightarrow Y_1 Y_2$ for $X \in N$ and $Y_1, Y_2 \in N$
    - $X \rightarrow Y$ for $X \in N$ and $Y \in \Sigma$
  - $S \in N$ is a distinguished start symbol

Cocke-Younger-Kasami Algorithm

- Notation:
  - n=number of words in the sentence
  - w_i=i-th word in the sentence (i.e. $s = w_1 \dots w_n$)
  - $\mathcal{T}(i, j, X)$ for $X \in N$ and $1 \le i \le j \le n$ is the set of all possible parse trees for words $w_i \dots w_j$ such that X is at the root of the tree
  - $\pi(i, j, X) = \max_{t \in \mathcal{T}(i,j,X)} p(t)$ i.e. $\pi(i, j, X)$ is the highest score for any parse tree in $\mathcal{T}(i, j, X)$
  - $\pi(1, n, S) = \max_{t \in \mathcal{T}_G(s)} p(t)$

Cocke-Younger-Kasami Algorithm

- Dynamic programmic parsing algorithm for PCFG in Chomsky Normal Form
- Bottom up approach in which $\pi(i, j, X)$ are recursively evaluated:
  - Base case ($i = j$):

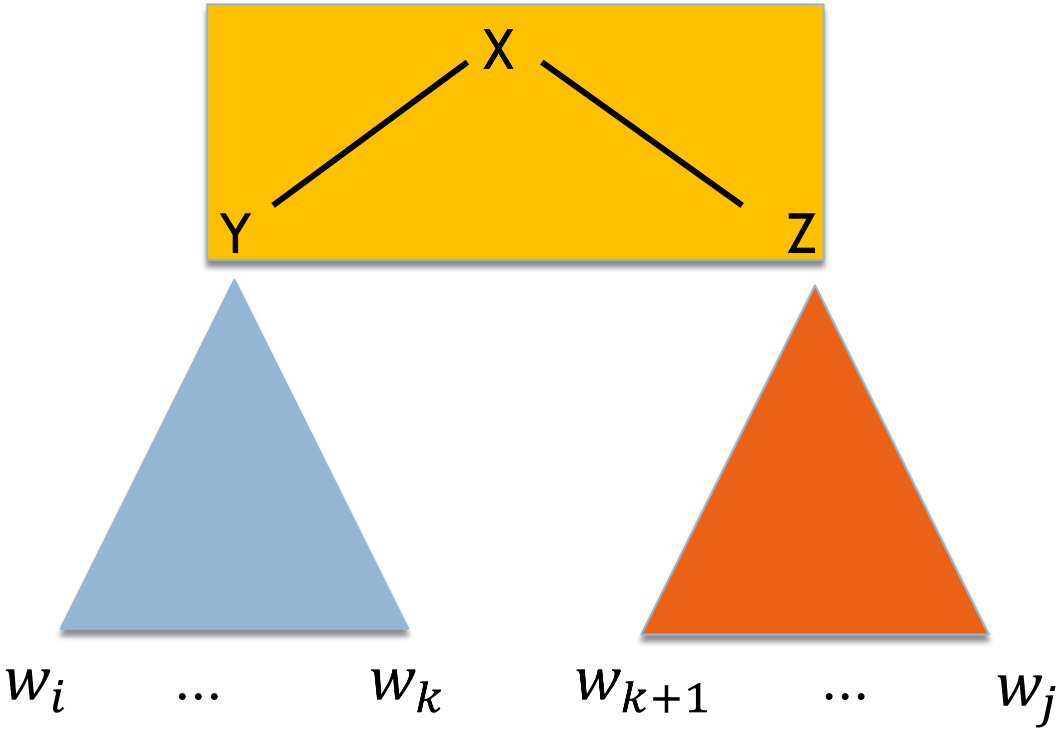
$$\pi(i, i, X) = \begin{cases} q(X \rightarrow \ x_i) & if\ X \rightarrow x_i \in R \\ 0 & otherwise \end{cases}$$

  - Recursive case ($i < j$):

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R \\ i \leq k \leq \ (j-1)}} \left(q(X \rightarrow YZ) \times \pi(i, k, Y) \times \pi(k+1, j, Z) \right)$$

Cocke-Younger-Kasami Algorithm

$$\pi(i,j,X) = \max_{\substack{X\to YZ\in R\\ i\leq k\leq (j-1)}} \left(q(X \to YZ) \times \pi(i,k,Y) \times \pi(k+1,j,Z) \right)$$



Cocke-Younger-Kasami Algorithm

- Input: a sentence $s = w_1 \dots w_n$ a PCFG G=(N, Σ, R, S, q)
- Initialization:

  For all $i \in \{1 \dots n\}$, for all $X \in N$

  $$\pi(i,i,X) = \begin{cases} q(X \to \; x_i) & if\ X \to x_i \in R \\ 0 & otherwise \end{cases}$$

- Algorithm:
  - For $l = 1 \dots (n-1)$
    - For $i = 1 \dots (n-l)$
      - Set $j = i + l$
      - For all $X \in N$ calculate

        $$\pi(i,j,X) = \max_{\substack{X \to YZ \in R \\ i \le k \le (j-1)}} \big(q(X \to YZ) \times \pi(i,k,Y) \times \pi(k+1,j,Z)\big)$$

        And store

        $$bp(i,j,X) = \operatorname*{argmax}_{\substack{X \to YZ \in R \\ i \le k \le (j-1)}} \big(q(X \to YZ) \times \pi(i,k,Y) \times \pi(k+1,j,Z)\big)$$

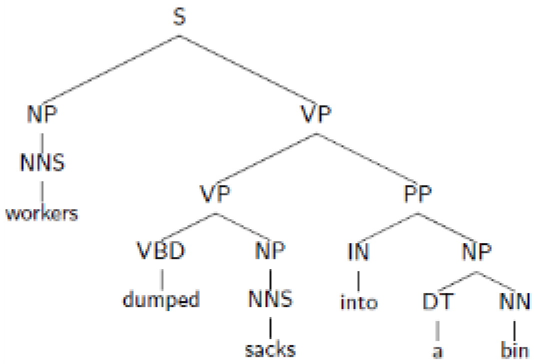
- Output: $bp(1,n,S)$

Overview

- Context Free Grammars
  - Ambiguity Problem
- **Probabilistic Context Free Grammars**
  - CYK parsing algorithm
  - **Weakness of PCFG**
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

Weakness of PCFG

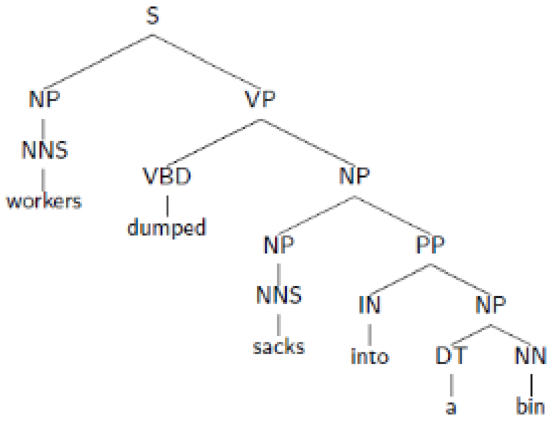
- **Lack of sensitivity to lexical information:** excluding the pre-terminal nodes (i.e. the Part-Of-Speeches) the probabilities $q(\alpha \rightarrow \ \beta)$ are completely independent of the words



Chosen if:
$q(NP \rightarrow NP\ PP) \leq q(VP \rightarrow VP\ PP)$

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

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



Chosen if:
$q(NP \rightarrow NP\ PP) > q(VP \rightarrow VP\ PP)$

Attachment decision is completely indipendent of the words

Weakness of PCFG

- **Lack of sensitivity to structural preferences:** the probabilities $q(\alpha \rightarrow \beta)$ focus only on α and β ignoring the overall tree structure

- For instance the sentence *John was believed to have been shot by Bill* can have at least two interpretations:
  - Bill does the shooting (the PP *by Bill* attaches to the verb *shot*)
  - Bill believes in John (the PP *by Bill* attaches to the verb *believe*)
- Both interpretations have the same rules and then identical probability
- Closer attachment should be preferred as a corpus analysis can demonstrate

Overview

- Context Free Grammars
  - Ambiguity Problem
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- **Lexicalized Context Free Grammars**
- Evaluation of parsing algorithms
- Statistical Dependency Parsing

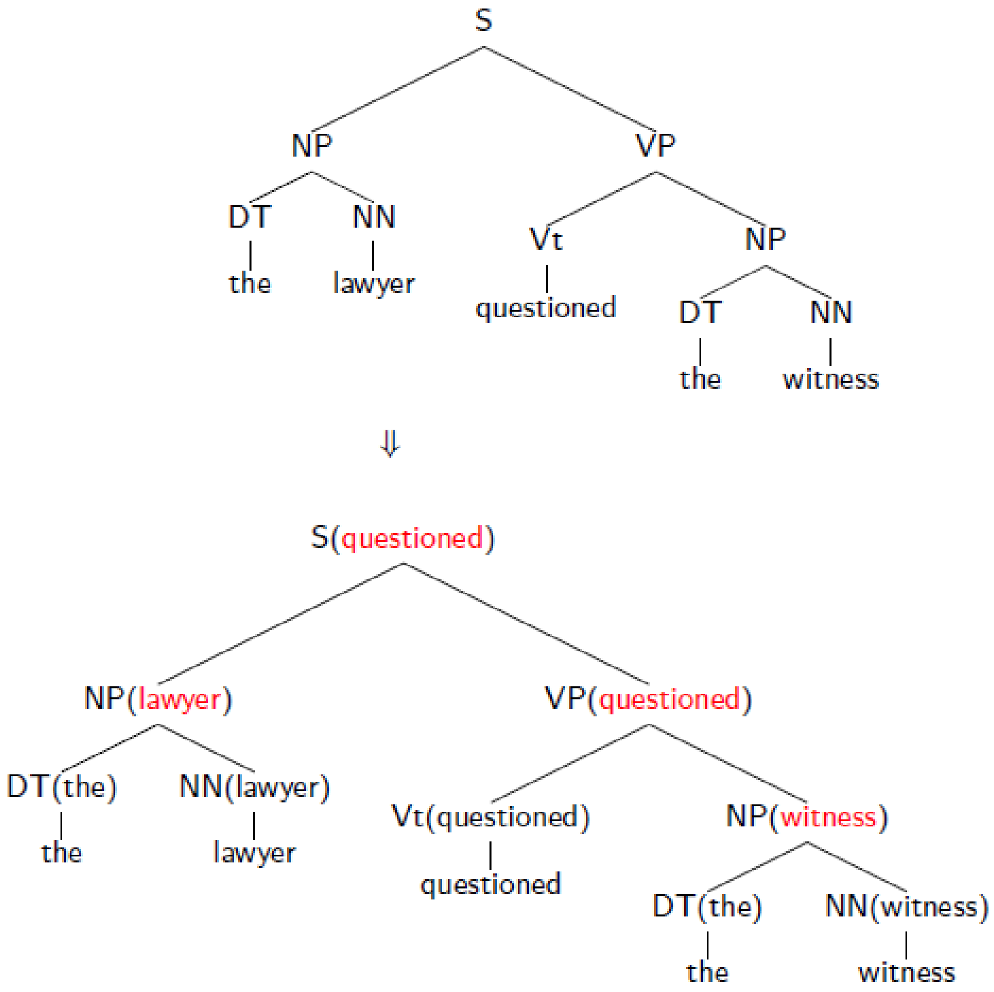
Lexicalization of a Treebank

- Idea: propagate the lexical information of the leaves (the words) through the entire tree

- According to some euristics, in each context-free rule a child is selected as *head* of the rule
  - S → NP VP (VP is the head)
  - VP → Vt NP (Vt is the head)
  - NP → DT NN NN (the last NN is the head)

- In a recursive bottom-up approach each constitient receives its headword from its head child

Adding Headwords to Trees



Lexicalized Context-Free Grammars in Chomsky Normal Form

- a Lexicalized Context-Free Grammar G=(*N*, Σ, *R*, *S*) in Chomsky Normal Form is as follow:
  - *N* is a set of non-terminal symbols
  - Σ is a set of terminal symbols
  - *R* is a set of rules which take one of three forms:
    - $X(h) \rightarrow_1 Y_1(h)Y_2(w)$ for $X \in N;\ \ Y_1, Y_2 \in N;\ \ h, w \in \Sigma$
    - $X(h) \rightarrow_2 Y_1(w)Y_2(h)$ for $X \in N;\ \ Y_1, Y_2 \in N;\ \ h, w \in \Sigma$
    - $X(h) \rightarrow h$ for $X \in N$ and $h \in \Sigma$
  - $S \in N$ is a distinguished start symbol

Lexicalized Context-Free Grammars

- The CYK algorithm is still valid but its *q* parameters have a different form:
  - An example of parameter in a PCFG:

$$q(S \rightarrow NP\ VP)$$

  - An example of parameter in a Lexicalized PCFG:

$$q(S(saw) \rightarrow_2 \ NP(man)\ VP(saw))$$

Parameter Estimation in Lexicalized PCFGs (Charniak 1997)

- First step: decompose a parameter into a product of two terms

$$q\big(S(saw) \rightarrow_2 NP(man)VP(saw)\big)$$
$$= q(S \rightarrow_2 NP\ VP|S, saw) \times q(man|S \rightarrow_2 NP\ VP, saw)$$

- Second Step: use smoothed estimation for the two term estimates

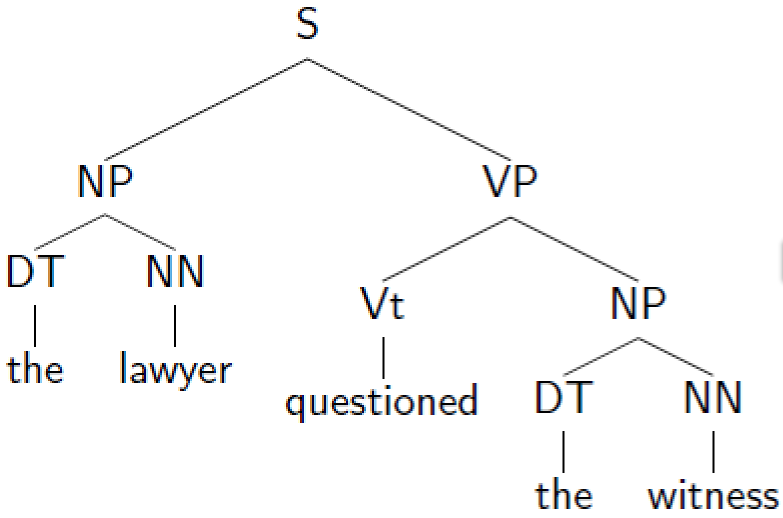
$$q(S \rightarrow_2 NP\ VP|S, saw)$$
$$= \lambda_1 \times q_{ML}(S \rightarrow_2 NP\ VP|S, saw) + (1 - \lambda_1) \times q_{ML}(S \rightarrow_2 NP\ VP|S)$$

$$q(man|S \rightarrow_2 NP\ VP, saw)$$
$$= \lambda_2 \times q_{ML}(man|S \rightarrow_2 NP\ VP, saw) + \lambda_3 \times q_{ML}(man|S \rightarrow_2 NP\ VP)$$
$$+ \lambda_4 \times q_{ML}(man|NP)$$

Overview

- Context Free Grammars
  - Ambiguity Problem
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- **Evaluation of parsing algorithms**
- Statistical Dependency Parsing

Evaluation: Representing Trees as Constituents



Label	Start Point	End Point
NP	1	2
NP	4	5
VP	3	5
S	1	5

Evaluation: Precision and Recall

Gold standard

Label	Start Point	End Point
NP	1	2
NP	4	5
NP	4	8
PP	6	8
NP	7	8
VP	3	8
S	1	8

Parse output

Label	Start Point	End Point
NP	1	2
NP	4	5
PP	6	8
NP	7	8
VP	3	8
S	1	8

- G = number of constituents in gold standard = 7
- P = number of constituents in parse output = 6
- C = number of correct constituents = 6

$$Recall = \frac{C}{G} = \frac{6}{7}$$

$$Precision = \frac{C}{P} = \frac{6}{6}$$

Some Results

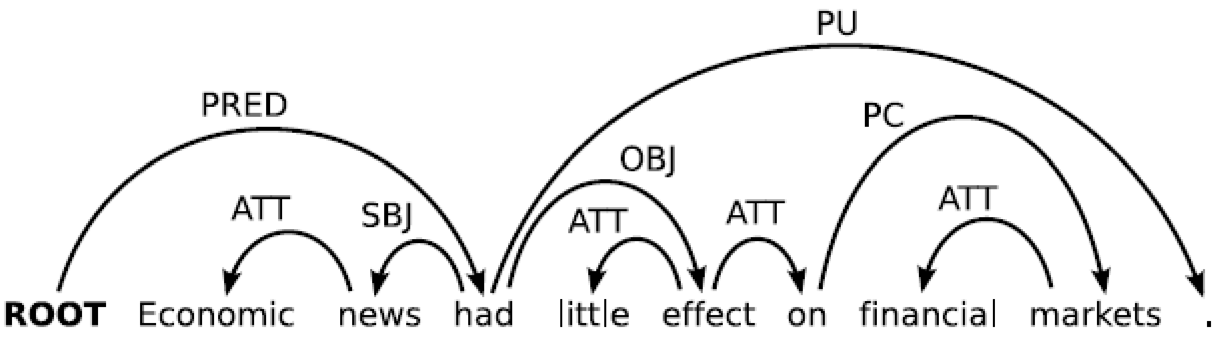
- Training data: 40,000 sentences from the Penn Wall Street Journal treebank. Testing: around 2,400 sentences from the Penn Wall Street Journal treebank
- Results for a PCFG: 70.6% Recall, 74.8% Precision
- Magerman (1994): 84.0% Recall, 84.3% Precision
- Results for a lexicalized PCFG: 88.1% recall, 88.3% precision (from Collins (1997, 2003))
- More recent results: 90.7% Recall/91.4% Precision (Carreras et al., 2008); 91.7% Recall, 92.0% Precision (Petrov 2010); 91.2% Recall, 91.8% Precision (Charniak and Johnson, 2005)

Overview

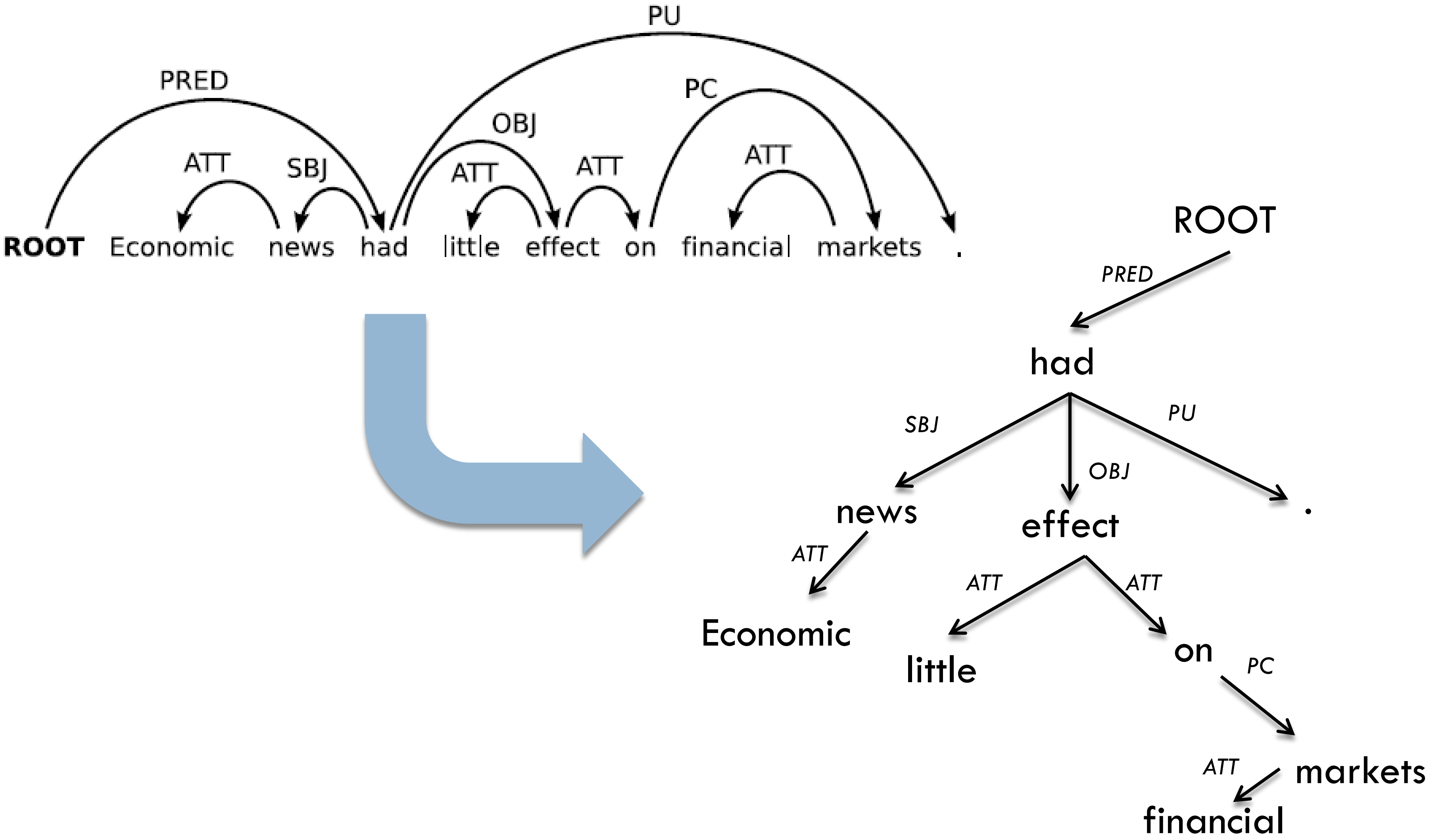
- Context Free Grammars
  - Ambiguity Problem
- Probabilistic Context Free Grammars
  - CYK parsing algorithm
  - Weakness of PCFG
- Lexicalized Context Free Grammars
- Evaluation of parsing algorithms
- **Statistical Dependency Parsing**

Dependency Parse Trees

- Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies
- The arrow connects a *head* with a *modifier* and are typed with the name of the grammatical relations



Dependency Parse Trees



Transition-Based Dependency Parsing

- Idea:
  - Define a transition system for dependency parsing
  - Learn a model for scoring possible transitions
  - Parse by searching for the optimal transition sequence
- Advantages:
  - Highly efficient parsing with low complexity
  - Rich history-based feature models for disambiguation

Transition System: Configurations

- Notation:
  - Arc (w_i, l, w_j) connects *head* w_i to modifier w_j with label *l*
  - Node w_0 (labeled ROOT) is the unique root of the tree
  - A configuration is a triple c=*(S,Q,A)* where:
    - *S* is a stack $[\ldots, w_i]_S$ of partially processed nodes
    - Q is a queue $\left[w_j, \ldots\right]_Q$ of remaining input nodes
    - *A* is a set of labeled arcs (w_i, l, w_j)
- Initialization:
  - $\left([w_0]_S, [w_1, \ldots, w_n]_Q, \{\}\right)$
- Termination:
  - $\left([w_0]_S, [\quad]_Q, A\right)$

Transition System: Transitions

- Three possible transitions:

$$\text{Left-Arc}(l)\ \frac{([\ldots, w_i, w_j]_S,\quad Q,\quad A)}{([\ldots, w_j]_S,\quad Q,\quad A \cup \{(w_j, l, w_i)\})}\ [i \neq 0]$$

$$\text{Right-Arc}(l)\ \frac{([\ldots, w_i, w_j]_S,\quad Q,\quad A)}{([\ldots, w_i]_S,\quad Q,\quad A \cup \{(w_i, l, w_j)\})}$$

$$\text{Shift}\ \frac{([\ldots]_S,\quad [w_i, \ldots]_Q,\quad A)}{([\ldots, w_i]_S,\quad [\ldots]_Q,\quad A)}$$

Transition Sequence Example

[ROOT]$_S$ [Economic, news, had, little, effect, on, financial, markets, .]$_Q$

ROOT Economic news had little effect on financial markets .

Transition Sequence Example

$[\text{ROOT, Economic}]_S$ $[\text{news, had, little, effect, on, financial, markets, .}]_Q$

ROOT Economic news had little effect on financial markets .

Transition Sequence Example

$[\textbf{ROOT}, \text{Economic}, \text{news}]_S$ $[\text{had}, \text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, .]_Q$

ROOT Economic news had little effect on financial markets .

Transition Sequence Example

$[\text{ROOT}, \text{news}]_S$ $[\text{had}, \text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, .]_Q$

Transition Sequence Example

$[\text{ROOT, news, had}]_S$ $[\text{little, effect, on, financial, markets, .}]_Q$

Transition Sequence Example

$[\textbf{ROOT}, \text{had}]_S$ $[\text{little}, \text{effect}, \text{on}, \text{financial}, \text{markets}, .]_Q$

Transition Sequence Example

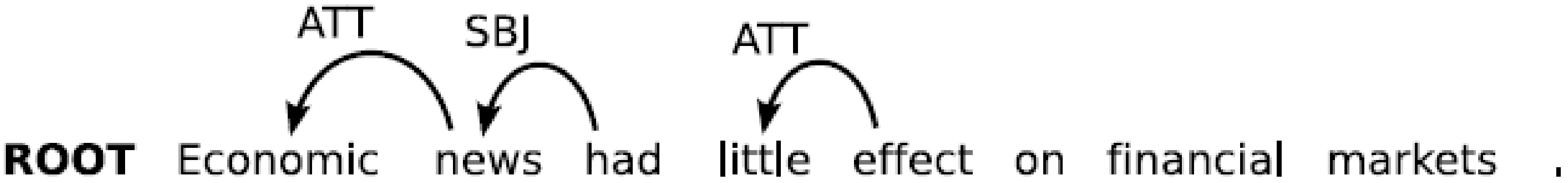
[**ROOT**, had, little]$_S$ [effect, on, financial, markets, .]$_Q$

Transition Sequence Example

[ROOT, had, little, effect]$_S$ [on, financial, markets, .]$_Q$

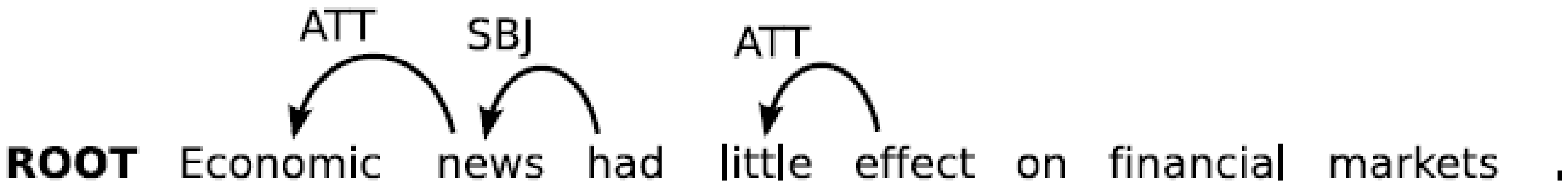
Transition Sequence Example

[**ROOT**, had, effect]$_S$ [on, financial, markets, .]$_Q$



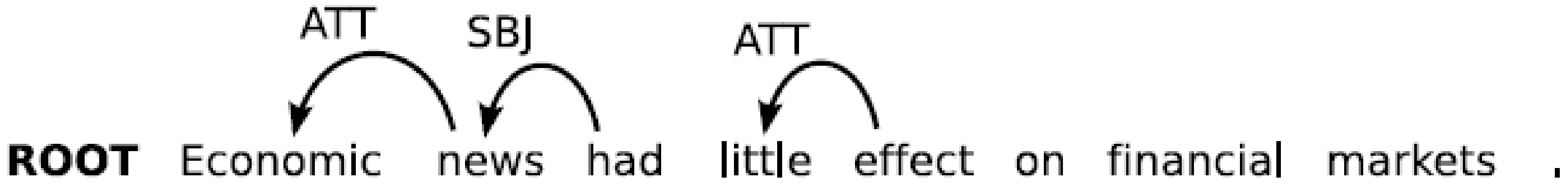
Transition Sequence Example

[ROOT, had, effect, on]$_S$ [financial, markets, .]$_Q$



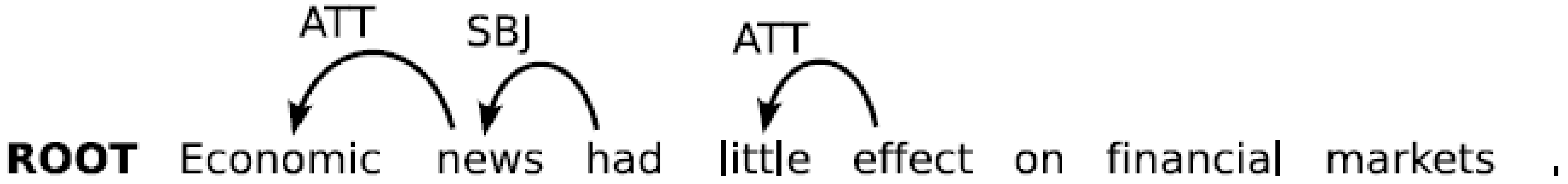
Transition Sequence Example

[**ROOT**, had, effect, on, financial]$_S$ [markets, .]$_Q$



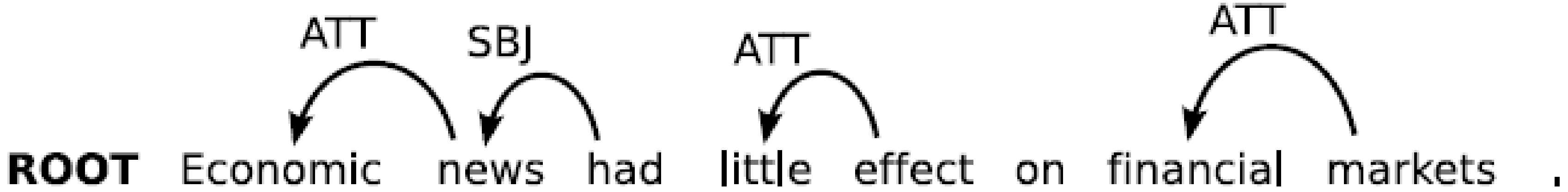
Transition Sequence Example

[ROOT, had, effect, on, financial, markets]$_S$ [.]$_Q$



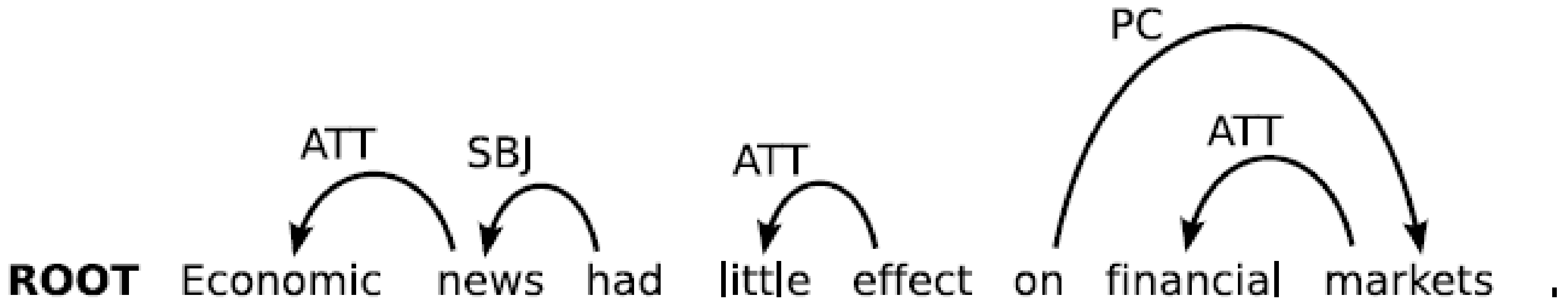
Transition Sequence Example

$[\text{ROOT}, \text{had}, \text{effect}, \text{on}, \text{markets}]_S \; [.]_Q$



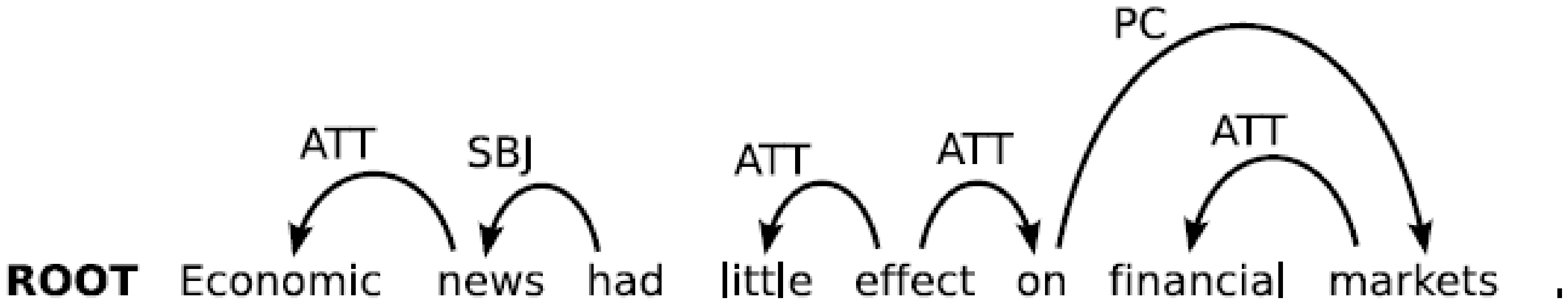
Transition Sequence Example

$[\text{ROOT}, \text{had}, \text{effect}, \text{on}]_S \ [.]_Q$



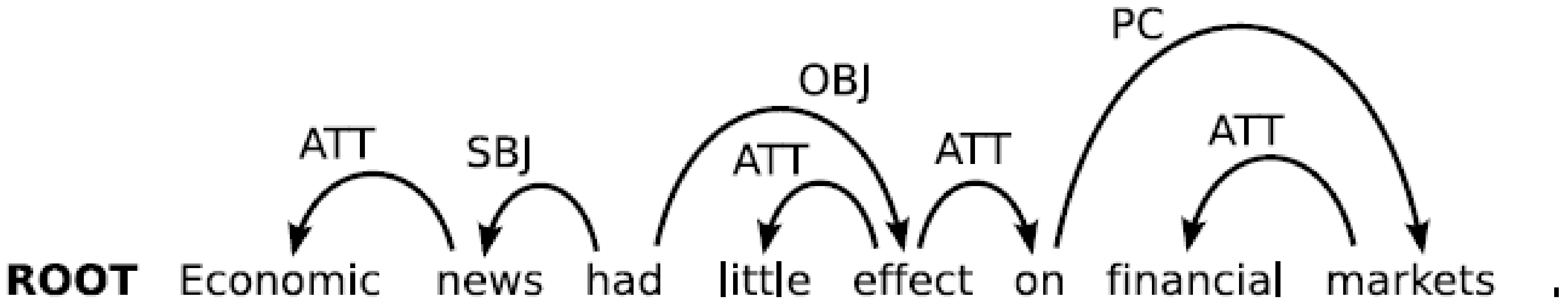
Transition Sequence Example

$[\text{ROOT}, \text{had}, \text{effect}]_S \ [.]_Q$



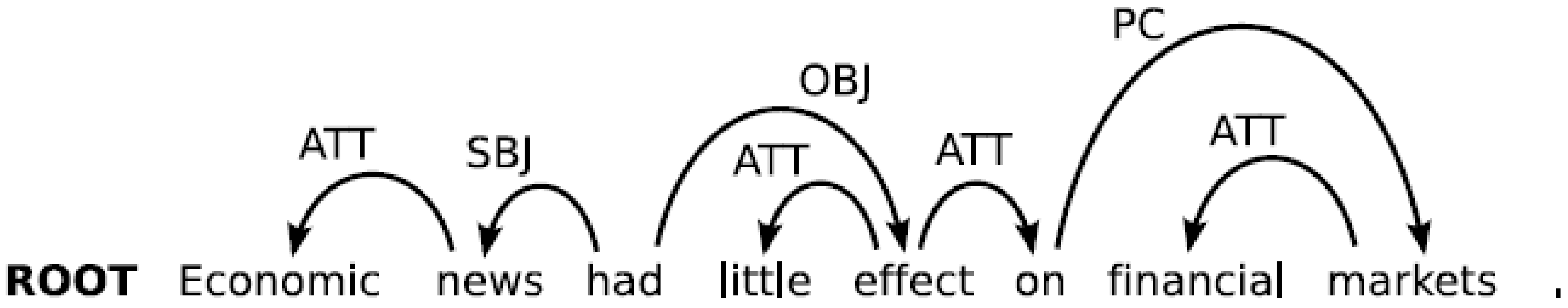
Transition Sequence Example

$[\text{ROOT}, \text{had}]_S \ [.]_Q$



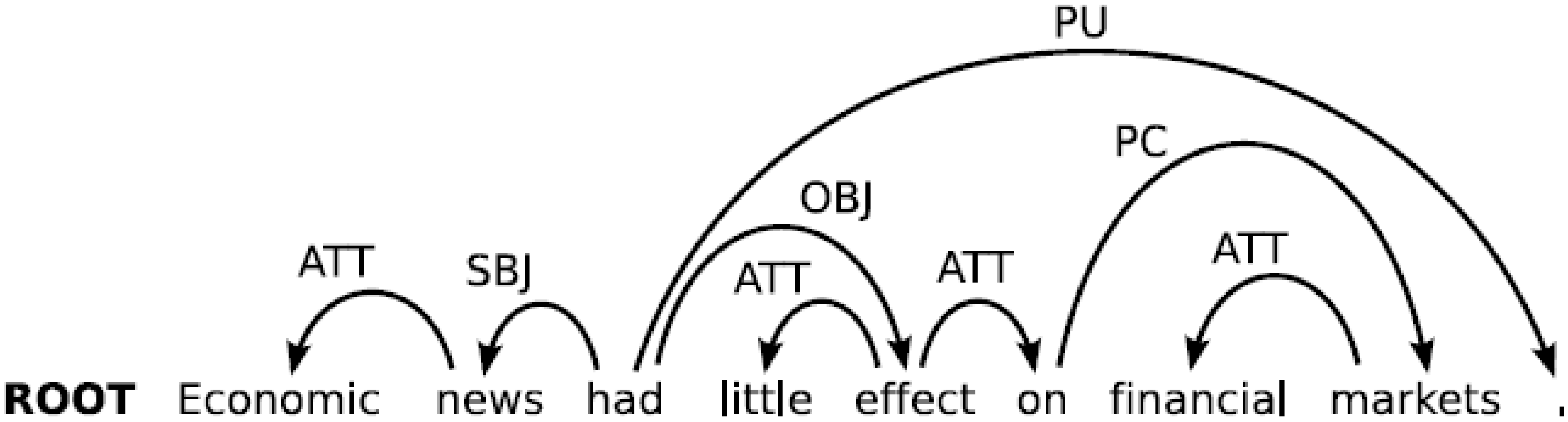
Transition Sequence Example

[ROOT, had, .]$_S$ [ ]$_Q$



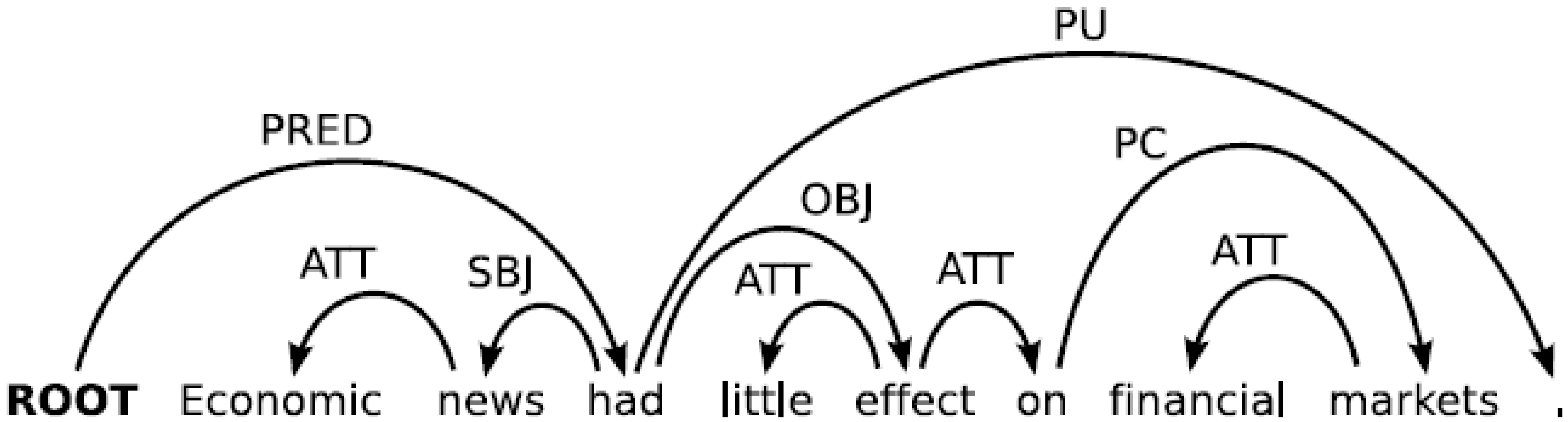
Transition Sequence Example

$[\text{ROOT}, \text{had}]_S \ [\ ]_Q$



Transition Sequence Example

$[\text{ROOT}]_S\ [\ ]_Q$



Selecting the Next Transition

- The next transition can be selected using a classifier (MaltParser):

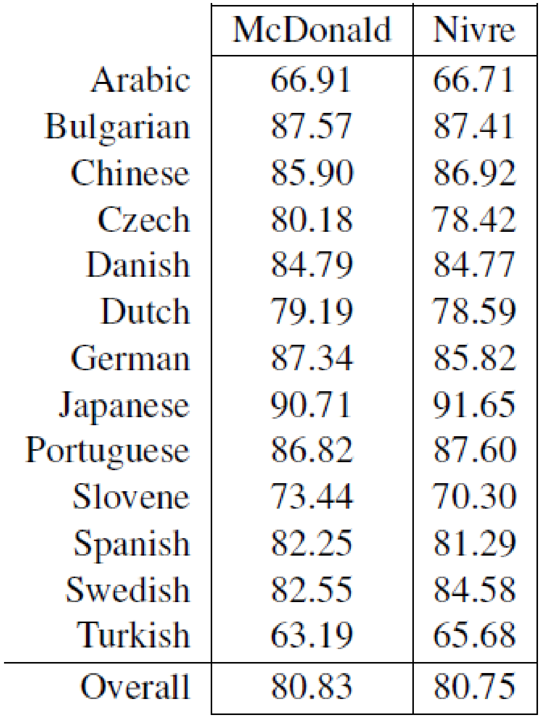
$$\text{Next transition} = \underset{t}{\operatorname{argmax}}\, \boldsymbol{w} \cdot \boldsymbol{f}(c, t)$$

- $\boldsymbol{f}(c, t)$ = Historic-based feature representation:
  - Features over input tokens relative to S and Q
  - Features over the (partial) dependency tree defined by A
  - Features over the (partial) transition sequence

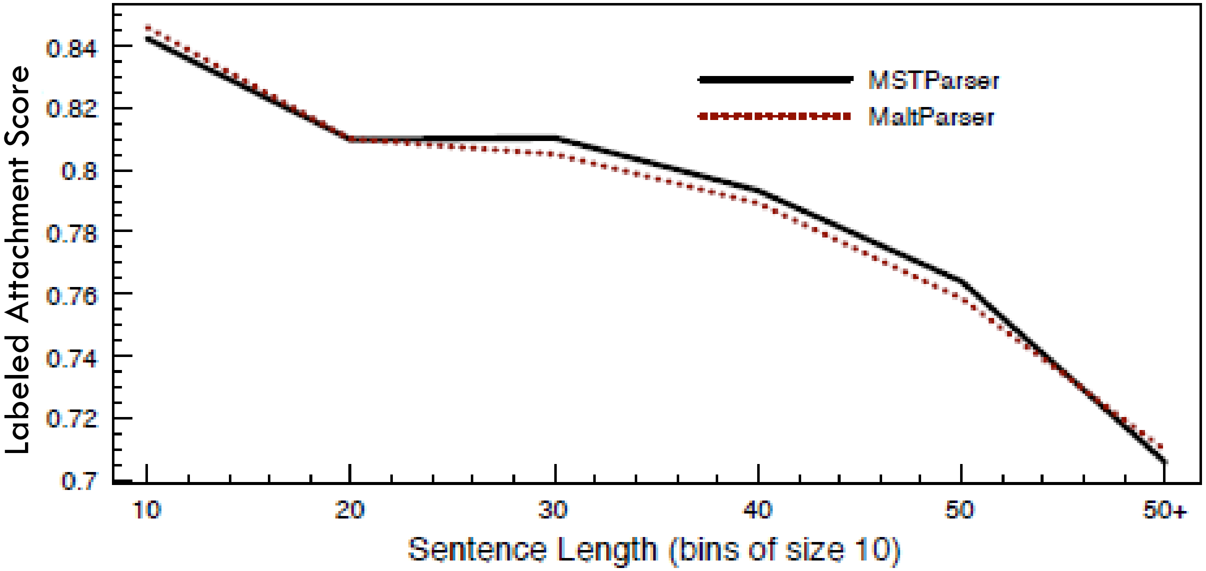
- $\boldsymbol{w}$ = weight vector learned from treebank data:
  - Reconstruct oracle transition sequence for Each sentence
  - Contruct training dataset $D = \{(c, t) | o(c) = t\}$
  - Maximize accuracy of local predictions

Evaluation of Dependency Parsing

- *Labeled Attachment Score*(LAS) = the percentage of tokens, excluding punctuation, that are assigned both the correct head and the correct dependency label
- *Unlabeled Attachment Score*(UAS) = the percentage of tokens, excluding punctuation, that are assigned the correct head



	McDonald	Nivre
Arabic	66.91	66.71
Bulgarian	87.57	87.41
Chinese	85.90	86.92
Czech	80.18	78.42
Danish	84.79	84.77
Dutch	79.19	78.59
German	87.34	85.82
Japanese	90.71	91.65
Portuguese	86.82	87.60
Slovene	73.44	70.30
Spanish	82.25	81.29
Swedish	82.55	84.58
Turkish	63.19	65.68
Overall	80.83	80.75



References

- Michael Collins:
  - Notes on PCFGs: http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf
  - Notes on Lexicalized PCFGs http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/lexpcfgs.pdf

- Joakim Nivre, Johan Hall and Jens Nilsson. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In Proc. of LREC 2006
- Ryan McDonald and Joakim Nivre. Characterizing the Errors of Data-Driven Dependency Parsing Models. In Proc. of EMNLP 2007