Natural Language Processing

Dependency Parsing

Joakim Nivre

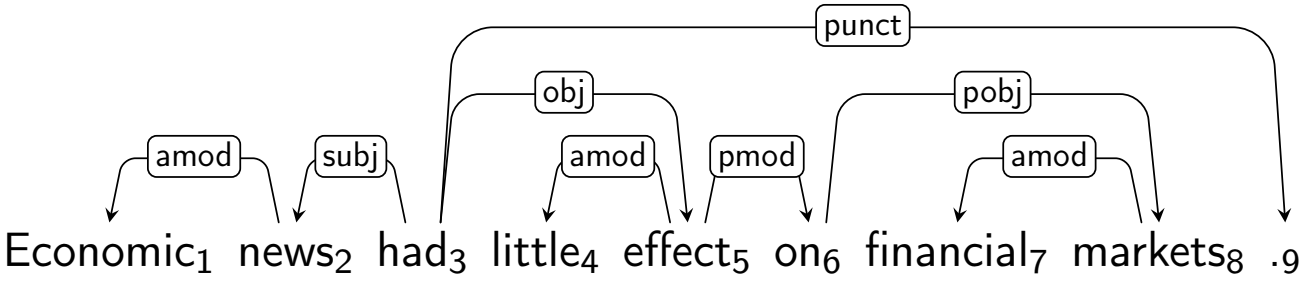
Uppsala University
Department of Linguistics and Philology
joakim.nivre@lingfil.uu.se

Syntactic Parsing

- Many NLP tasks need to analyse sentence structure
  - Question answering – Which company acquired Nokia?
  - Information extraction – Microsoft bought Nokia
- Syntactic parsing:
  - **Input:** sentence (sequence of words)
  - **Output:** syntactic representation
- Dependency parsing – parsing to dependency trees

Dependency Trees

- A dependency tree is a directed graph $G = (V, A)$, where
  - the nodes in V are the words $w_1, \ldots, w_n$ of a sentence
  - the arcs in A have the form (w_i, l, w_j), where l is label

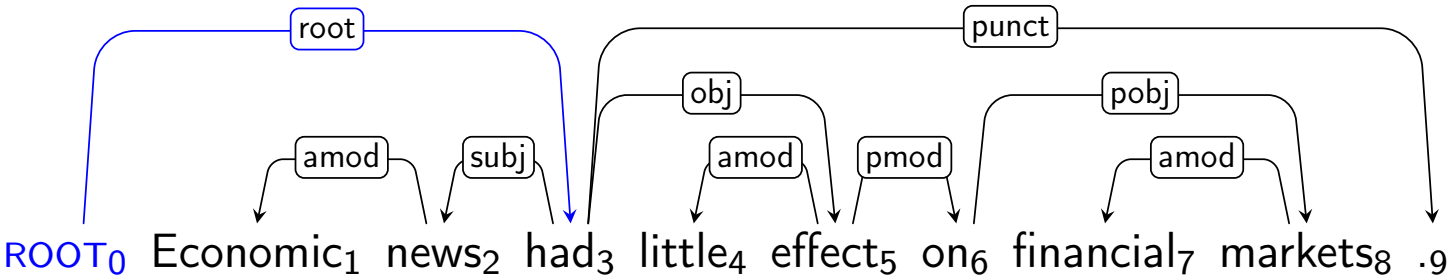


$V =$ {Economic$_1$, news$_2$, had$_3$, little$_4$, effect$_5$, on$_6$, financial$_7$, markets$_8$, .$_9$}

$A =$ {(news$_2$, amod, Economic$_1$), (had$_3$, subj, news$_2$), (effect$_5$, amod, little$_4$), (had$_3$, obj, news$_5$), (effect$_5$, pmod, on$_6$), (markets$_8$, amod, financial$_7$), (on$_6$, pobj, markets$_8$), (had$_3$, punct, .$_9$) }

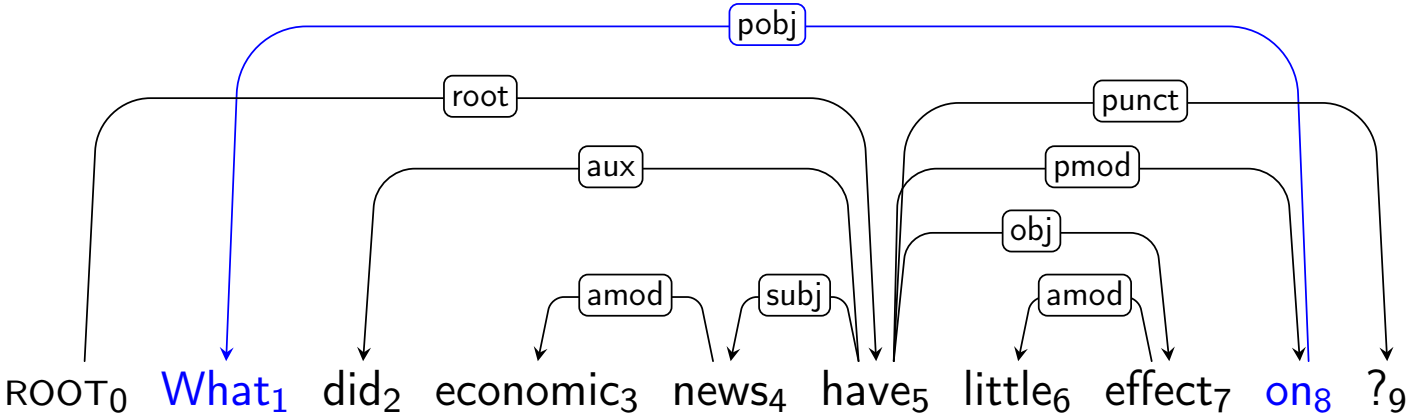
Dependency Trees

- A graph $G = (V, A)$ is a well-formed tree if and only if:
  - every node has at most one incoming arc (single head)
  - the graph is (weakly) connected (no dangling nodes)
  - there are no cycles (no word is dependent on itself)
- For convenience, we often add a dummy root word
  - then every node has exactly one incoming arc (single head)
  - and connectedness is easily satisfied

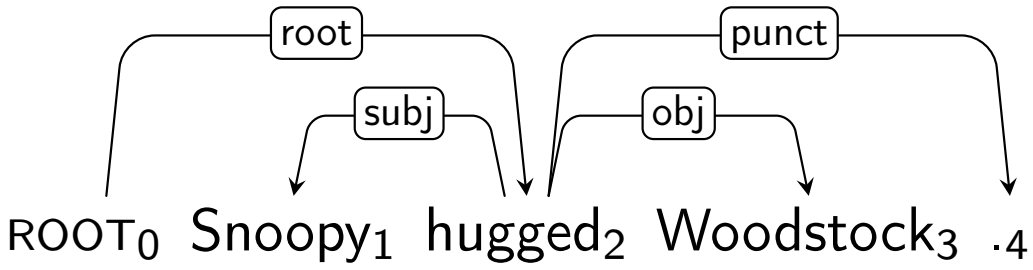


Projectivity

- A dependency tree is projective if there are no discontinuities
  - all words between a head w_i and a dependent w_j must be reachable from w_i
- Non-projective trees may be needed to account for
  - long-distance dependencies
  - free word order



Quiz



- Which of the following arcs are in the tree above?
  1. (ROOT$_0$, root, hugged$_2$)
  2. (hugged$_2$, subj, Snoopy$_1$)
  3. (hugged$_2$, subj, Woodstock$_3$)
  4. (Snoopy$_1$, obj, Woodstock$_3$)

Dependency Parsing

- The problem:
  - Input: Sentence $S = w_0, w_1, \ldots, w_n$ ($w_0 = \text{ROOT}$)
  - Output: Dependency tree $G = (V, A)$ for S
- Data-driven dependency parsing
  - Learn a model with the help of a treebank $\mathcal{T}$:
    - where $\mathcal{T} = \{(S_1, G_1), (S_2, G_2), \ldots, (S_n, G_n)\}$
    - where each (S_i, G_i) is a sentence and its dependency tree
  - Parse by finding the best tree according to the model

Two Approaches

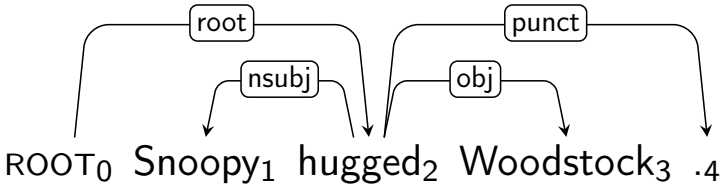
- Graph-based parsing:
  - Define a space of candidate dependency trees for a sentence
  - Learn to score dependency trees, factored into subgraphs
  - Parse by finding the highest-scoring dependency tree
- Transition-based parsing:
  - Define a set of actions to build a dependency tree
  - Learn to predict the next action, given the history
  - Parse by finding the optimal sequence of actions

Evaluation Metrics

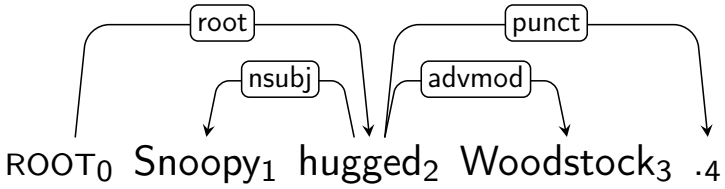
- Standard setup
  - Test set $\mathcal{E} = \{(S_1, G_1), (S_2, G_2), \ldots, (S_n, G_n)\}$
  - Parser predictions $\mathcal{P} = \{(S_1, G'_1), (S_2, G'_2), \ldots, (S_n, G'_n)\}$
- Evaluation metrics
  - Labeled attachment score (LAS) =
    Percentage of words that get the correct head and label
  - Unlabeled attachment score (UAS) =
    Percentage of words that get the correct head

Quiz

Test set tree



Parser output



- True or false?
  1. The labeled attachment score (LAS) is 100%
  2. The unlabeled attachment score (LAS) is 100%
  3. The labeled attachment score (LAS) is 75%
  4. The unlabeled attachment score (LAS) is 75%

Natural Language Processing

Transition-Based Dependency Parsing

Joakim Nivre

Uppsala University
Department of Linguistics and Philology
joakim.nivre@lingfil.uu.se

The Basic Idea

- Define a set of actions to build a dependency tree
- Learn to predict the next action, given the history
- Parse by finding the optimal sequence of actions

The Basic Idea

- Define a set of actions to build a dependency tree
- Learn to predict the next action, given the history
- Parse by finding the optimal sequence of actions

Parsing actions are defined by a transition system

Transition System: Configurations

- A parser configuration is a triple $c = (S, B, A)$, where
  - S = a stack $[\ldots, w_i]_S$ of partially processed nodes,
  - B = a buffer $[w_j, \ldots]_B$ of remaining input nodes,
  - A = a set of labeled arcs (w_i, l, w_j).

Transition System: Configurations

- A parser configuration is a triple $c = (S, B, A)$, where
  - S = a stack $[\ldots, w_i]_S$ of partially processed nodes,
  - B = a buffer $[w_j, \ldots]_B$ of remaining input nodes,
  - A = a set of labeled arcs (w_i, l, w_j).
- Initialization:

$$([w_0]_S, [w_1, \ldots, w_n]_B, \{\,\})$$

NB: $w_0 = \text{ROOT}$

Transition System: Configurations

- A parser configuration is a triple $c = (S, B, A)$, where
  - S = a stack $[\ldots, w_i]_S$ of partially processed nodes,
  - B = a buffer $[w_j, \ldots]_B$ of remaining input nodes,
  - A = a set of labeled arcs (w_i, l, w_j).
- Initialization:

  $$([w_0]_S, [w_1, \ldots, w_n]_B, \{\,\})$$

  NB: $w_0 = \text{ROOT}$
- Termination:

  $$(S, [\,]_B, A)$$

Transition System: Transitions

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

Transition System: Transitions

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

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

Transition System: Transitions

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

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

- Reduce $\dfrac{([\ldots, w_i]_S, \quad B, \quad A)}{([\ldots]_S, \quad B, \quad A)}\ \exists k, l : (w_k, l, w_i) \in A$

Transition System: Transitions

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

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

- Reduce $\dfrac{([\ldots, w_i]_S, \quad B, \quad A)}{([\ldots]_S, \quad B, \quad A)}$ $\exists k, l : (w_k, l, w_i) \in A$

- Shift $\dfrac{([\ldots]_S, \quad [w_i, \ldots]_B, \quad A)}{([\ldots, w_i]_S, \quad [\ldots]_B, \quad A)}$

Example Transition Sequence

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

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	ADJ	NOUN	VERB	ADJ	NOUN	ADP	ADJ	NOUN	PUNCT

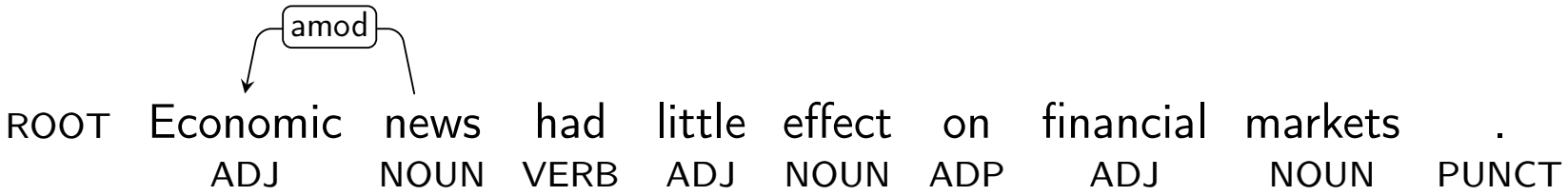
Example Transition Sequence

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

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	ADJ	NOUN	VERB	ADJ	NOUN	ADP	ADJ	NOUN	PUNCT

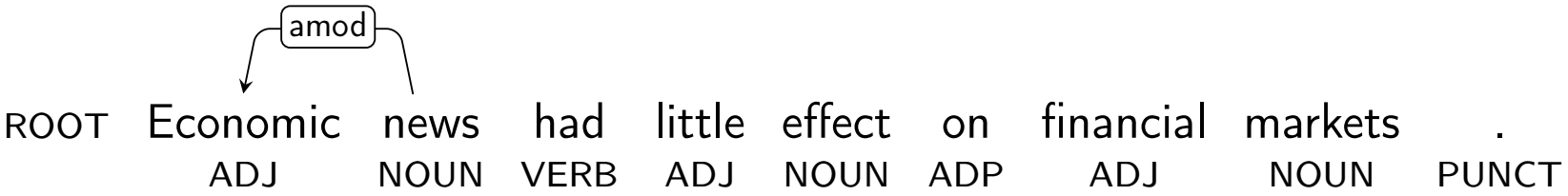
Example Transition Sequence

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



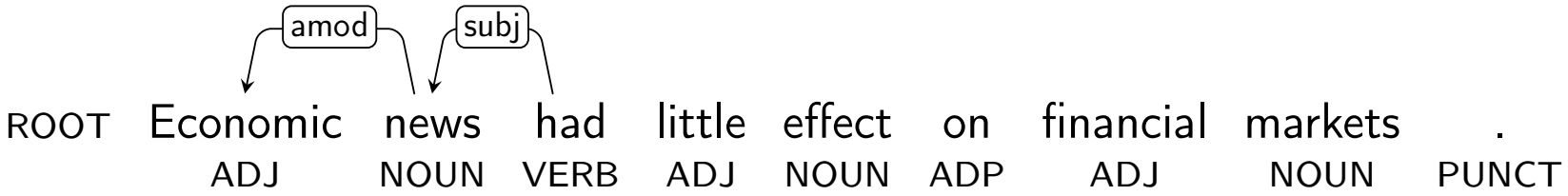
Example Transition Sequence

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



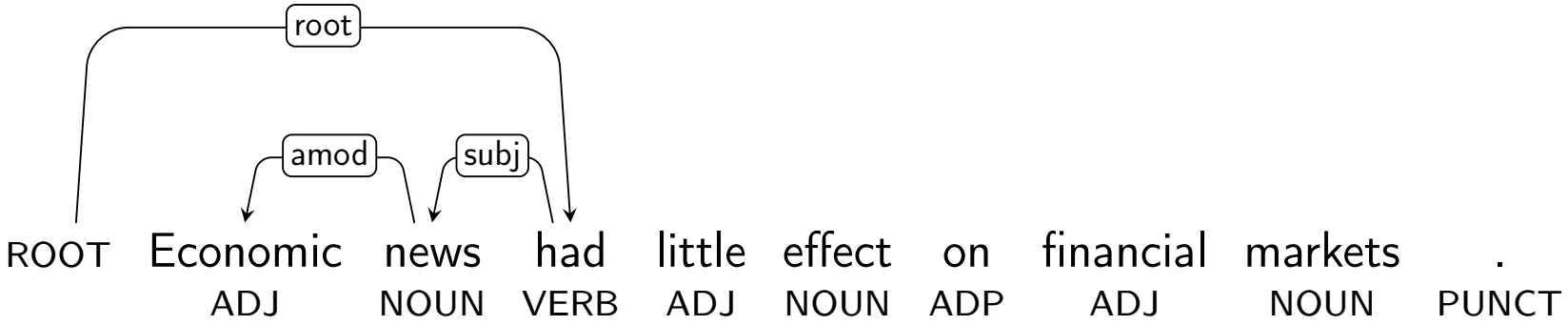
Example Transition Sequence

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



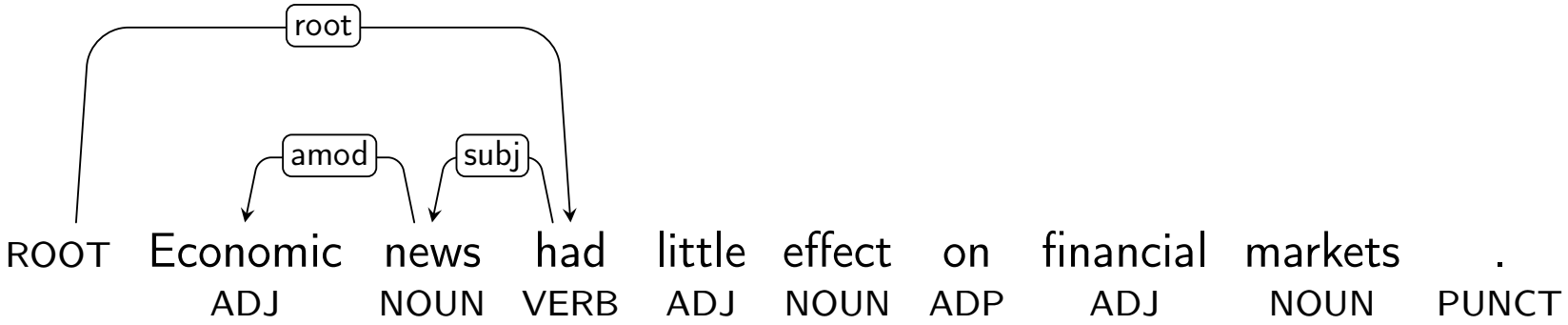
Example Transition Sequence

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



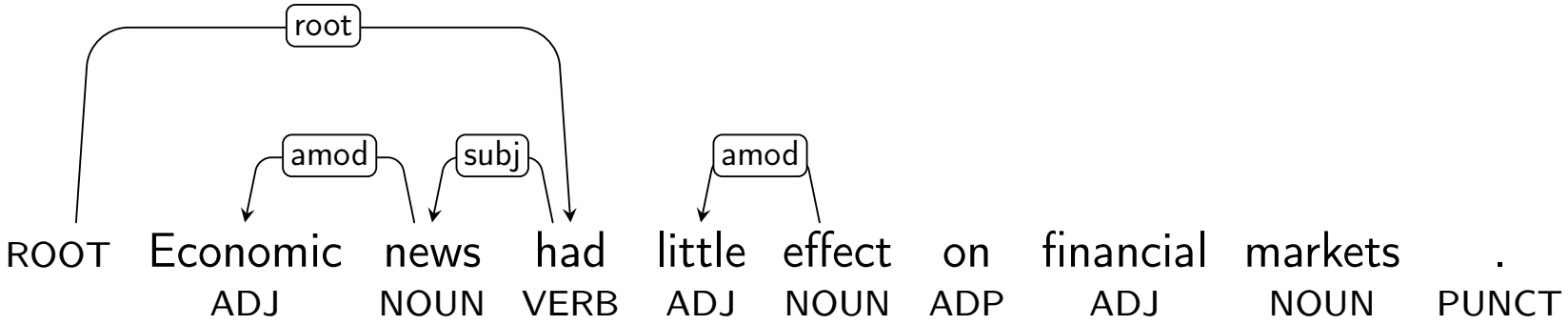
Example Transition Sequence

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



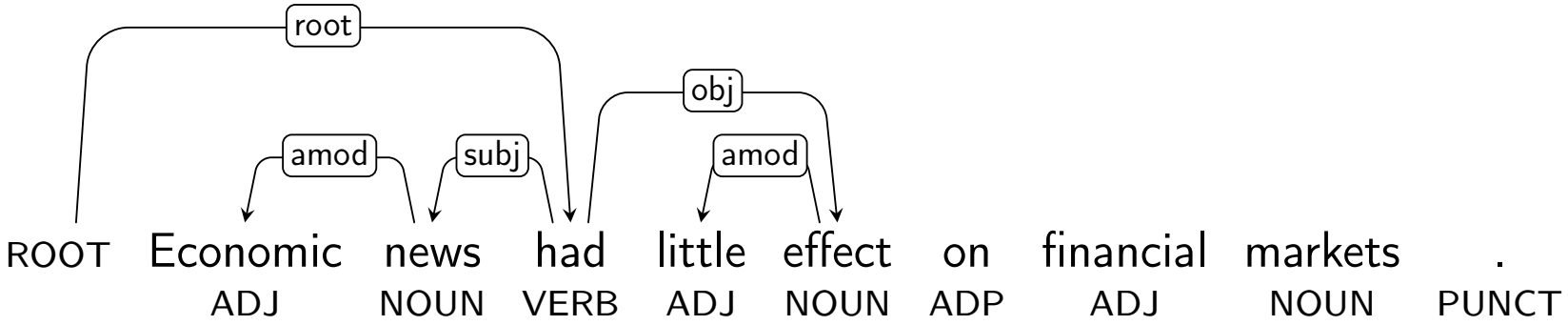
Example Transition Sequence

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



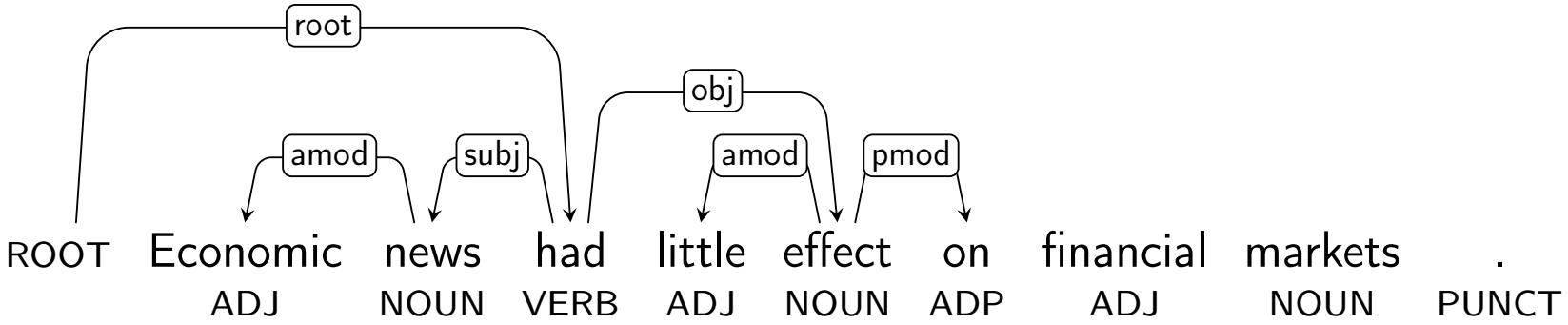
Example Transition Sequence

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



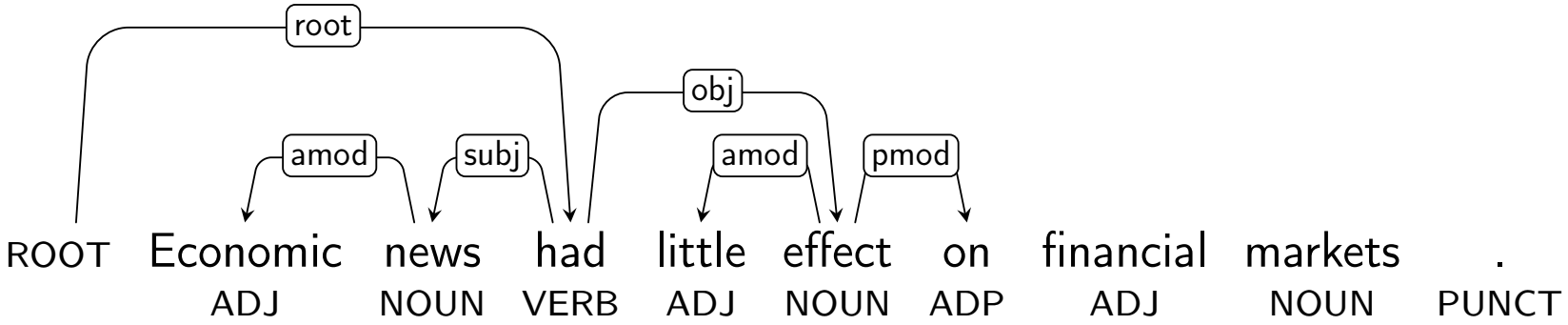
Example Transition Sequence

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



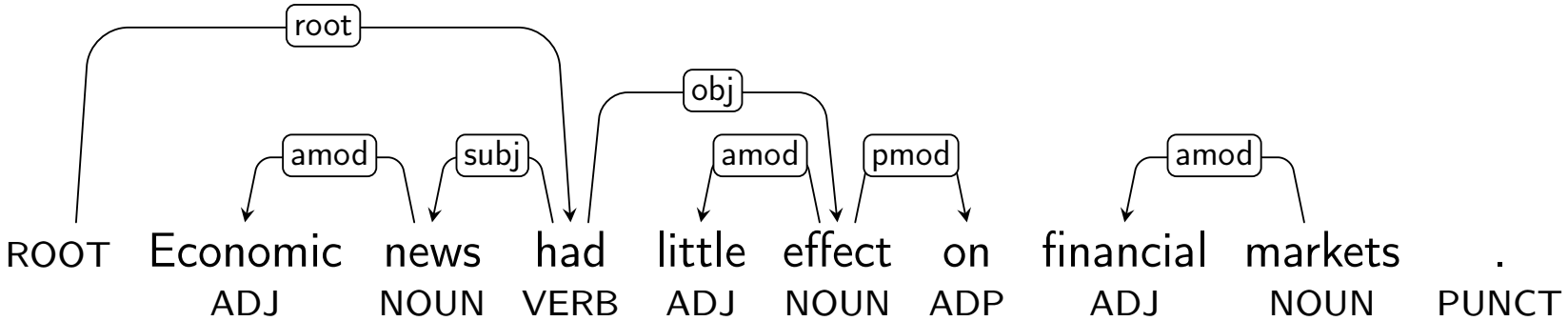
Example Transition Sequence

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



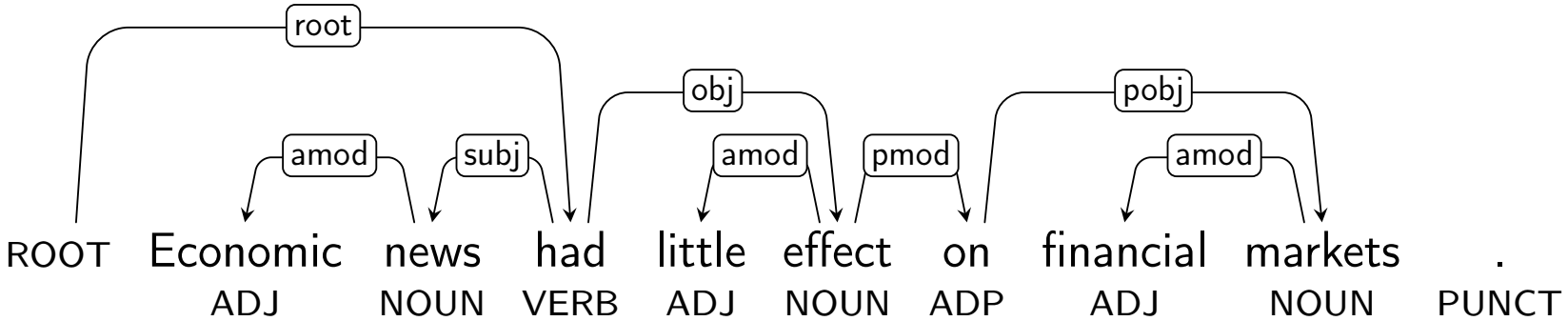
Example Transition Sequence

[ROOT, had, effect, on]$_S$ [markets, .]$_B$



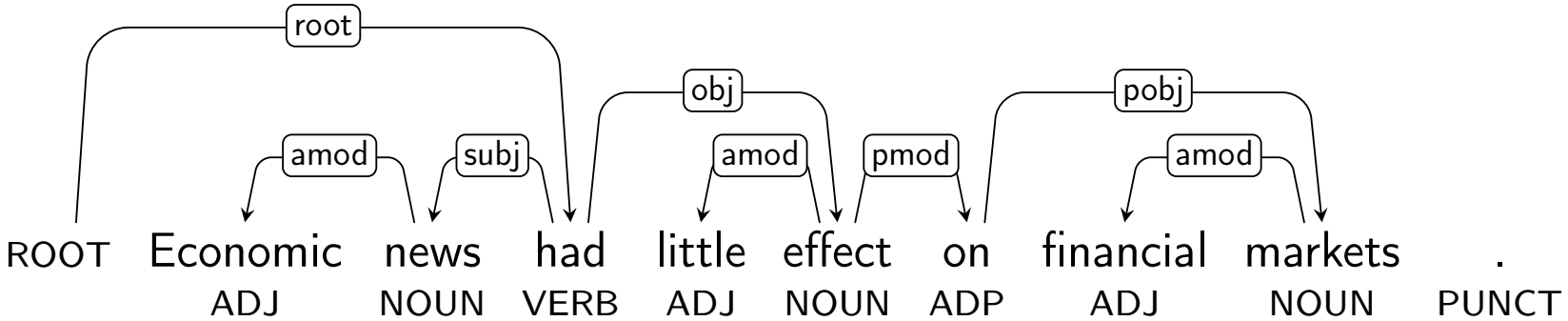
Example Transition Sequence

[ROOT, had, effect, on, markets]$_S$ [.]$_B$



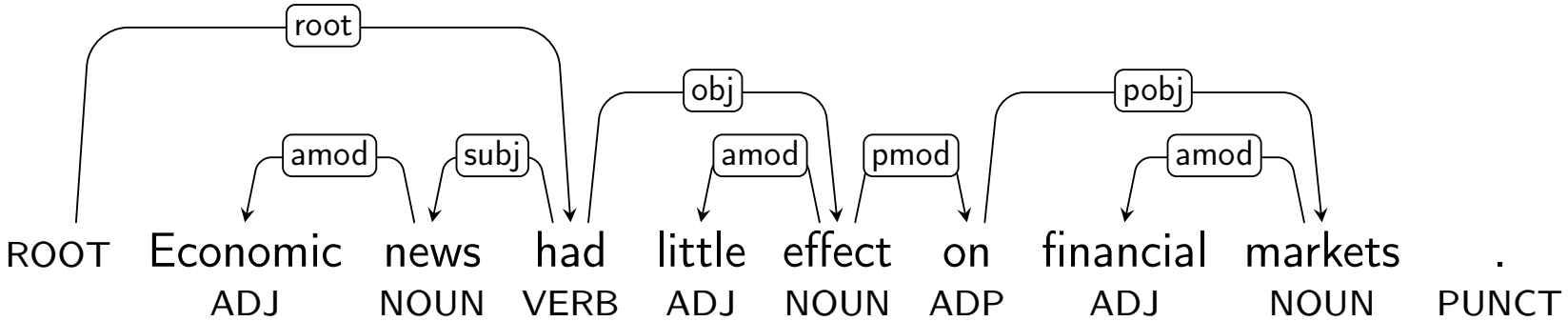
Example Transition Sequence

[ROOT, had, effect, on]$_S$ [.]$_B$



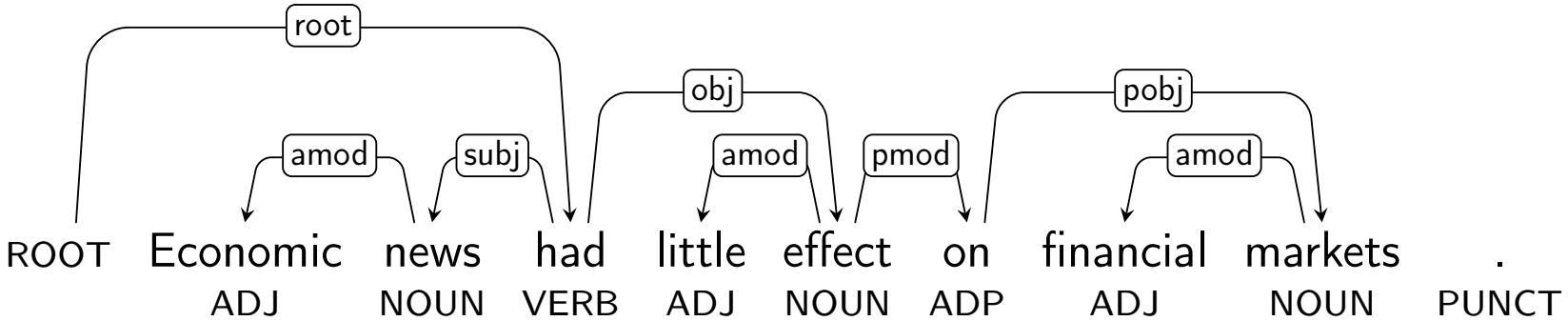
Example Transition Sequence

[ROOT, had, effect]$_S$ [.]$_B$



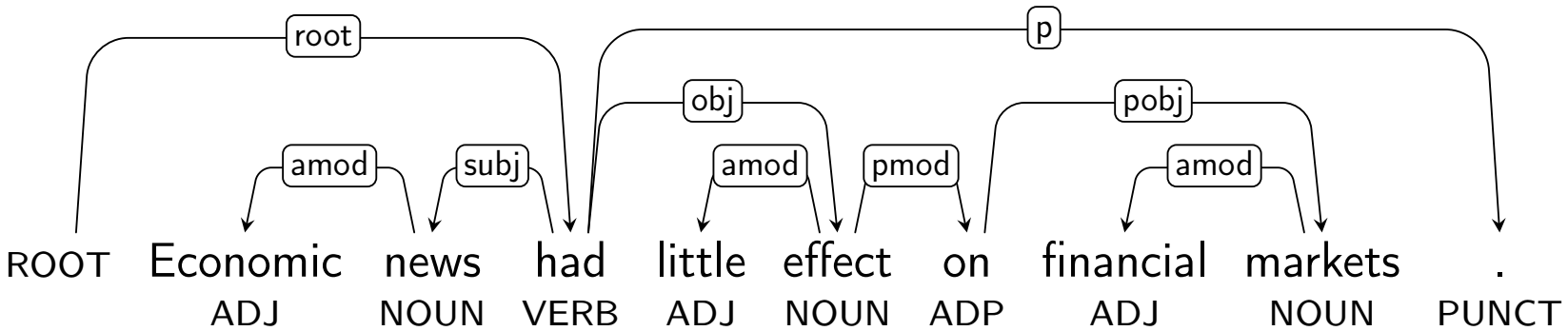
Example Transition Sequence

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



Example Transition Sequence

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



Quiz

$[\text{ROOT}, \text{had}, \text{effect}, \text{on}, \text{financial}]_S \quad [\text{markets}, .]_B$

- What happens if we make a Left-Arc transition here?
  1. The word markets is added to the stack S
  2. The word financial is removed from the stack S
  3. An arc is added to the arc set A
  4. An arc is removed from the arc set A

Parsing Algorithm

- Given an oracle o that correctly predicts the next transition $o(c)$, parsing is deterministic:

$$
\begin{array}{l}
\text{Parse}(w_0, w_1, \ldots, w_n) \\
\quad c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_B, \{\ \}) \\
\quad \textbf{while } B_c \neq [\ ] \\
\qquad\quad t \leftarrow o(c) \\
\qquad\quad c \leftarrow t(c) \\
\quad \textbf{return } G = (\{w_0, w_1, \ldots, w_n\}, A_c)
\end{array}
$$

Parsing Algorithm

- Given an oracle o that correctly predicts the next transition $o(c)$, parsing is deterministic:

$$
\begin{array}{l}
\text{Parse}(w_0, w_1, \ldots, w_n) \\
\quad c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_B, \{\ \}) \\
\quad \textbf{while } B_c \neq [\ ] \\
\qquad\quad t \leftarrow o(c) \\
\qquad\quad c \leftarrow t(c) \\
\quad \textbf{return } G = (\{w_0, w_1, \ldots, w_n\}, A_c)
\end{array}
$$

Parsing time is linear in sentence length

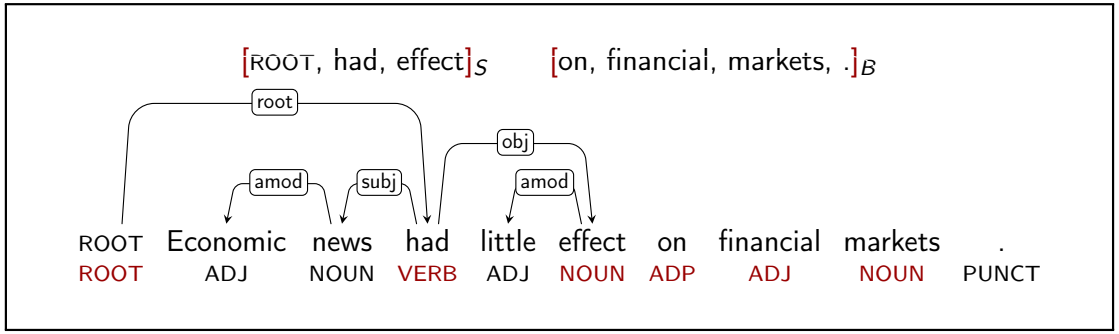
Oracles and Machine Learning

- We can approximate oracles using machine learning
- Given a treebank:
  - We know the dependency tree for each sentence
  - We know the transitions $t_1, \ldots, t_n$ we need to build the tree
  - We can learn to imitate the oracle by predicting each t_i from c_i
- We represent each configuration by a finite set of features

Features

- Features over input tokens relative to S and B

Configuration



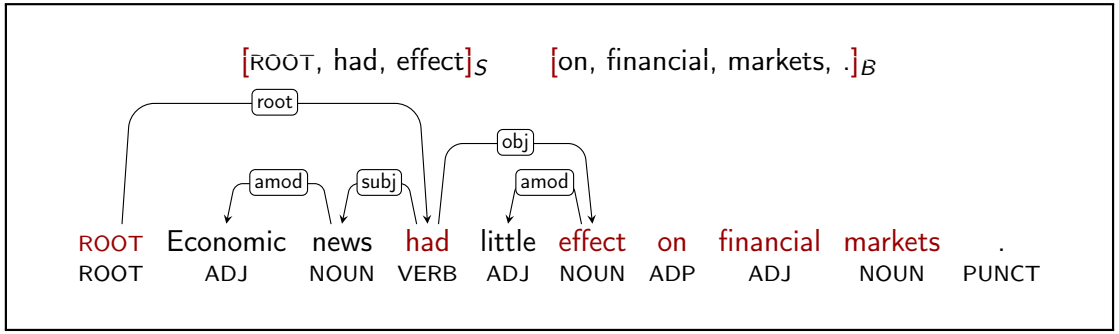
Features

$$
\begin{aligned}
\text{pos}(S_2) &= \text{ROOT} \\
\text{pos}(S_1) &= \text{VERB} \\
\text{pos}(S_0) &= \text{NOUN} \\
\text{pos}(B_0) &= \text{ADP} \\
\text{pos}(B_1) &= \text{ADJ} \\
\text{pos}(B_2) &= \text{NOUN}
\end{aligned}
$$

Features

- Features over input tokens relative to S and B

Configuration



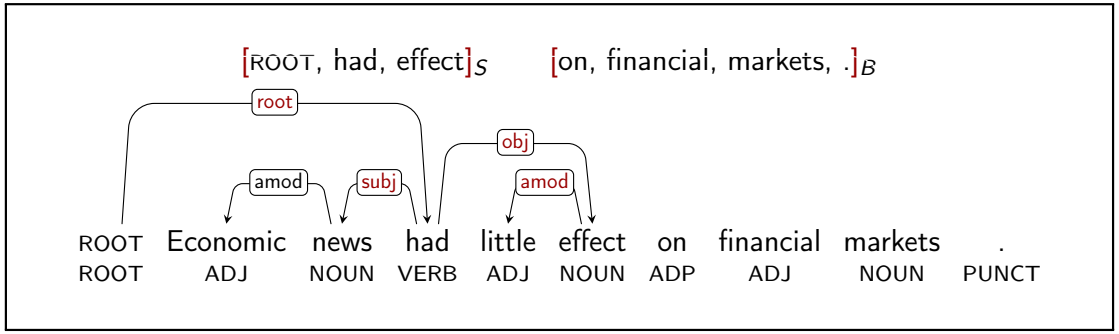
Features

$$
\begin{aligned}
\text{word}(S_2) &= \text{ROOT} \\
\text{word}(S_1) &= \text{had} \\
\text{word}(S_0) &= \text{effect} \\
\text{word}(B_0) &= \text{on} \\
\text{word}(B_1) &= \text{financial} \\
\text{word}(B_2) &= \text{markets}
\end{aligned}
$$

Features

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A

Configuration



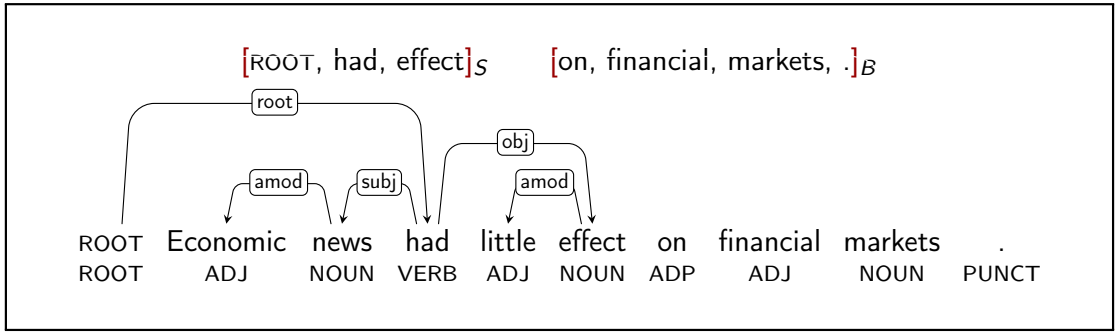
Features

$$
\begin{aligned}
\text{dep}(S_1) &= \text{root} \\
\text{dep}(\text{lc}(S_1)) &= \text{subj} \\
\text{dep}(\text{rc}(S_1)) &= \text{obj} \\
\text{dep}(S_0) &= \text{obj} \\
\text{dep}(\text{lc}(S_0) &= \text{amod} \\
\text{dep}(\text{rc}(S_0) &= \text{NIL}
\end{aligned}
$$

Features

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A
- Features over the (partial) transition sequence

Configuration



Features

t_{i-1} = Right-Arc(obj)
t_{i-2} = Left-Arc(amod)
t_{i-3} = Shift
t_{i-4} = Right-Arc(root)
t_{i-5} = Left-Arc(subj)
t_{i-6} = Shift

Transition-Based Parsing

- Advantages:
  - Very efficient parsing
  - Rich feature representations
- Drawback:
  - Sensitive to prediction errors and error propagation

Non-Projective Parsing

- The arc-eager transition system can only build projective trees

Non-Projective Parsing

- The arc-eager transition system can only build projective trees
- We can add arcs between non-adjacent words:
  - Left-Arc-NP(l)

$$\frac{([\ldots, w_i, w_k]_S, \quad [w_j, \ldots]_B, \quad A)}{([\ldots, w_k]_S, \quad [w_j, \ldots]_B, \quad A \cup \{(w_j, l, w_i)\})}$$

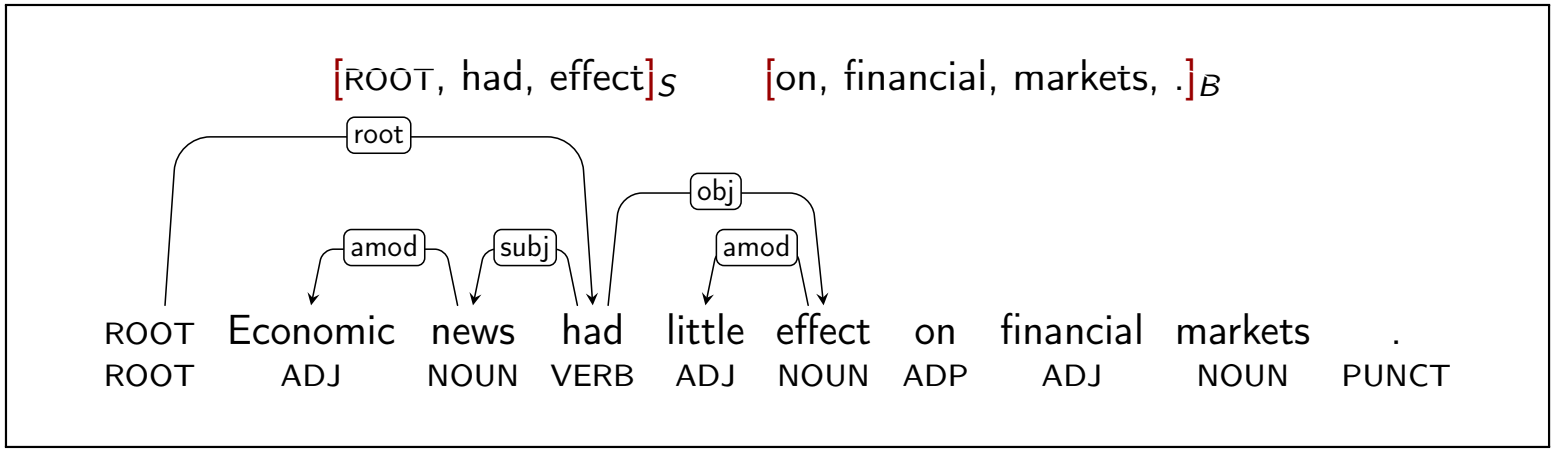
Non-Projective Parsing

- The arc-eager transition system can only build projective trees
- We can add arcs between non-adjacent words:
  - Left-Arc-NP(l)

$$\frac{([\ldots, w_i, w_k]_S, \quad [w_j, \ldots]_B, \quad A)}{([\ldots, w_k]_S, \quad [w_j, \ldots]_B, \quad A \cup \{(w_j, l, w_i)\})}$$

- Or we can reorder words:
  - Swap $\dfrac{([\ldots, w_i]_S, \quad [w_j, \ldots]_B, \quad A)}{([\ldots]_S, \quad [w_j, w_i, \ldots]_B, \quad A)}$

Quiz



- True or false in the configuration above?
  1. $\text{pos}(S_0) = \text{pos}(B_0)$
  2. $\text{pos}(S_0) = \text{pos}(B_1)$
  3. $\text{pos}(S_0) = \text{pos}(B_2)$
  4. $\text{pos}(S_0) = \text{pos}(B_3)$

Natural Language Processing

Graph-Based Dependency Parsing

Joakim Nivre

Uppsala University
Department of Linguistics and Philology
joakim.nivre@lingfil.uu.se

The Basic Idea

- Define a space of candidate dependency trees for a sentence
- Learn to score dependency trees, factored into subgraphs
- Parse by finding the highest-scoring dependency tree

The Basic Idea

- Define a space of candidate dependency trees for a sentence
- Learn to score dependency trees, factored into subgraphs
- Parse by finding the highest-scoring dependency tree

We score trees, not parsing actions

Arc-Factored Models

- The score of a tree is the sum of the score of its subgraphs:

$$s(G) = \sum_{i=1}^{n} s(G_i)$$

- In the arc-factored model, every arc is a separate subgraph:

$$s(G = (V, A)) = \sum_{(w_i, l, w_j) \in A} s(w_i, l, w_j)$$

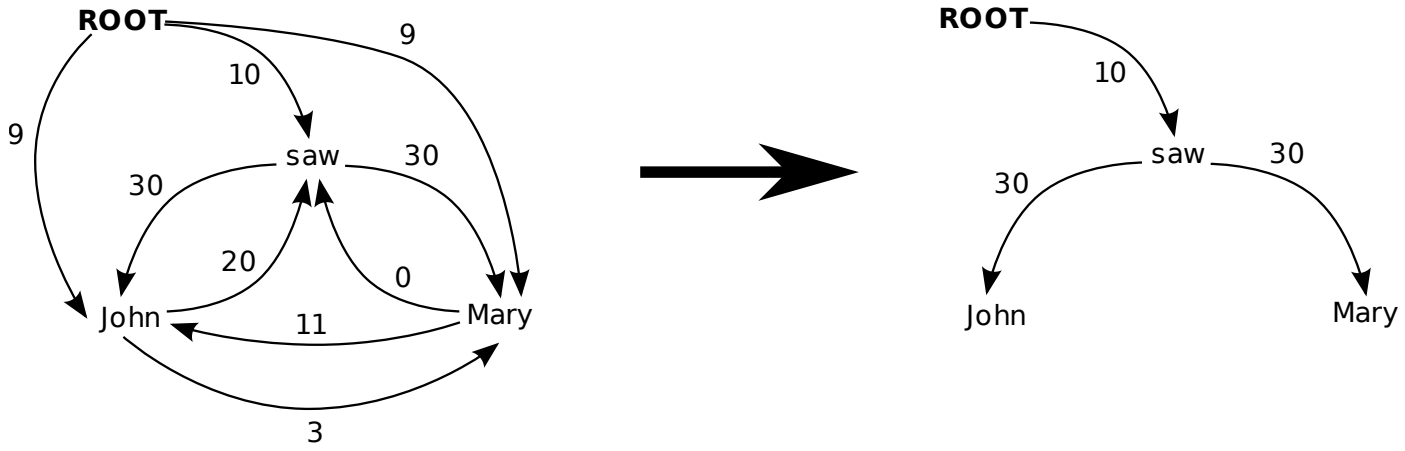
- We can simplify to unlabeled trees (and add labels later):

$$s(G = (V, A)) = \sum_{(w_i, w_j) \in A} s(w_i, w_j)$$

Parsing

- Given arc scores, a parser should find the highest scoring tree

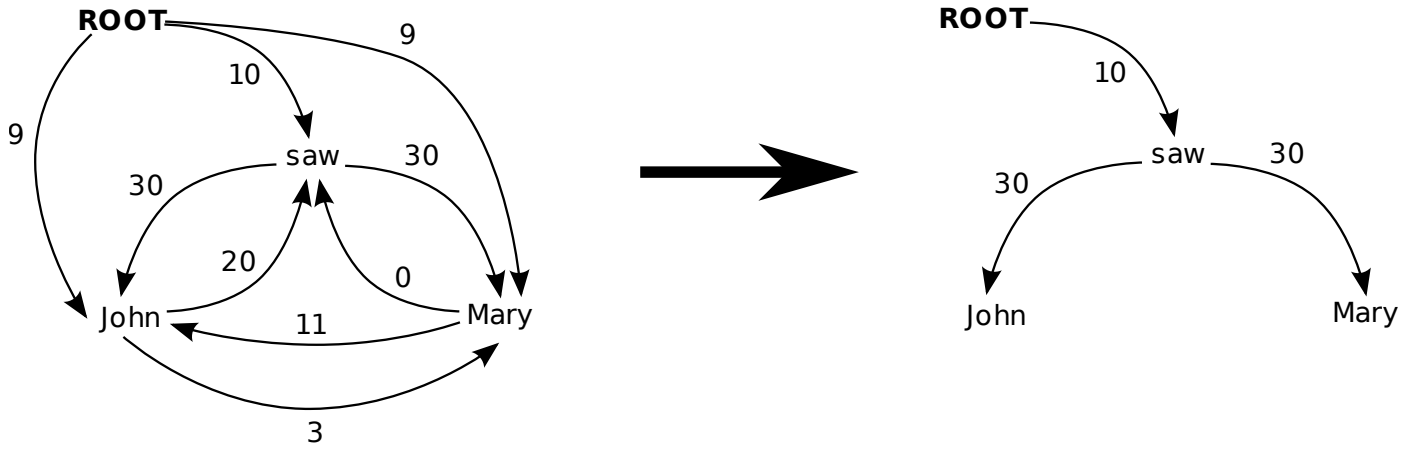
$$G* = \underset{G=(V,A)}{\text{argmax}} \sum_{(w_i,w_j)\in A} s(w_i, w_j)$$



Parsing

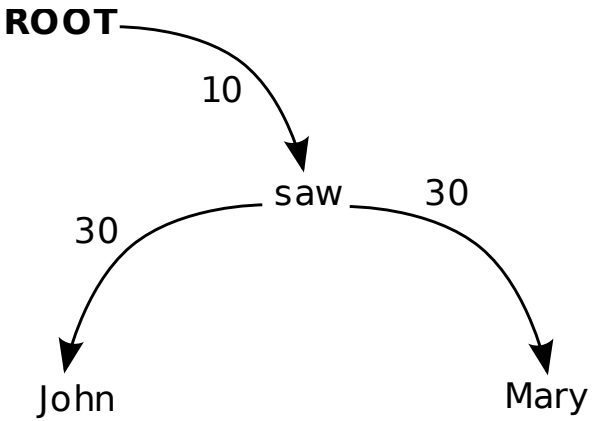
- Given arc scores, a parser should find the highest scoring tree

$$G* = \underset{G=(V,A)}{\operatorname{argmax}} \sum_{(w_i,w_j)\in A} s(w_i, w_j)$$



This is not the same as the highest-scoring head for each word!

Quiz



- What is the score of this tree under the arc-factored model?
  1. 10
  2. 30
  3. 70
  4. 9000

Dynamic Programming

- Naive approach:
  - Enumerate all possible trees and score them one by one
  - Number of trees grows exponentially with sentence length
  - But many trees are composed of the same substructures
- Smarter approach:
  - Avoid scoring the same substructure more than once
  - Score smaller substructures before larger
  - Keep a table over scored substructures

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

for span size s from 2 to $n+1$ # from smaller to larger

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

for span size s from 2 to $n+1$ # from smaller to larger
  for start position i from 0 to $n+1-s$ # from left to right

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

for span size s from 2 to $n+1$ # from smaller to larger
  for start position i from 0 to $n+1-s$ # from left to right
    find best split j ($i < j < i+s$), head w_h and dependent w_d

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

for span size s from 2 to $n+1$ # from smaller to larger
  for start position i from 0 to $n+1-s$ # from left to right
    find best split j ($i < j < i+s$), head w_h and dependent w_d
      if $h < j \leq d$ **then** # head in left subtree
        $S[i][i+s][h] := S[i][j][h] + S[j][i+s][d] + s(w_h, w_d)$
        $A[i][i+s][h] := A[i][j][h] \cup A[j][i+s][d] \cup \{(w_h, w_d)\}$

Parsing Algorithm

0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

Parse table: $S[i][j][h]$ = score of the best subtree from i to j with head w_h
$A[i][j][h]$ = arcs of the best subtree from i to j with head w_h

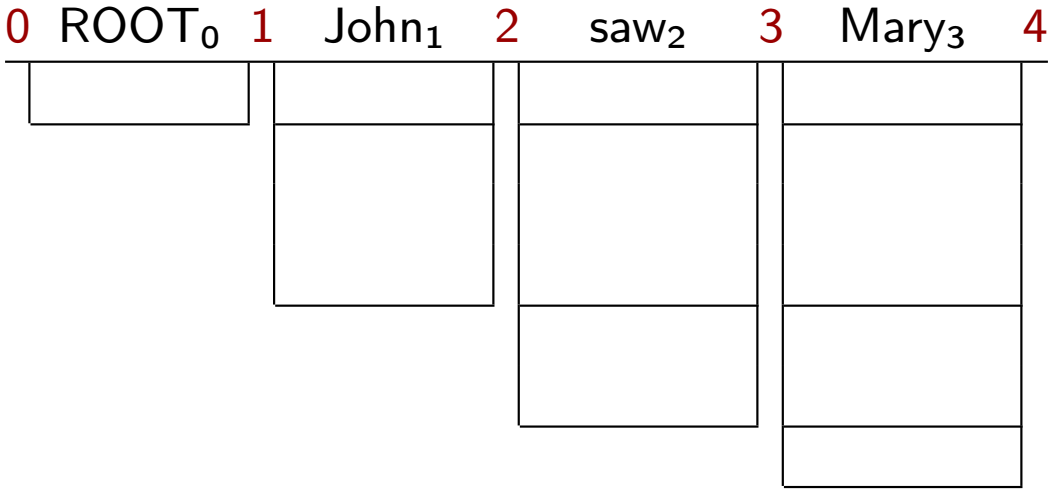
Best tree: $A[0][n+1][h]$ (for some h)

Initialize: $S[i][i+1][i] = 0$ (for all i from 0 to n)
$A[i][i+1][i] = \emptyset$ (for all i from 0 to n)

for span size s from 2 to $n+1$ # from smaller to larger
  for start position i from 0 to $n+1-s$ # from left to right
    find best split j ($i < j < i+s$), head w_h and dependent w_d
      if $h < j \leq d$ **then** # head in left subtree
        $S[i][i+s][h] := S[i][j][h] + S[j][i+s][d] + s(w_h, w_d)$
        $A[i][i+s][h] := A[i][j][h] \cup A[j][i+s][d] \cup \{(w_h, w_d)\}$
      else # head in right subtree
        $S[i][i+s][h] := S[i][j][d] + S[j][i+s][h] + s(w_h, w_d)$
        $A[i][i+s][h] := A[i][j][d] \cup A[j][i+s][h] \cup \{(w_h, w_d)\}$

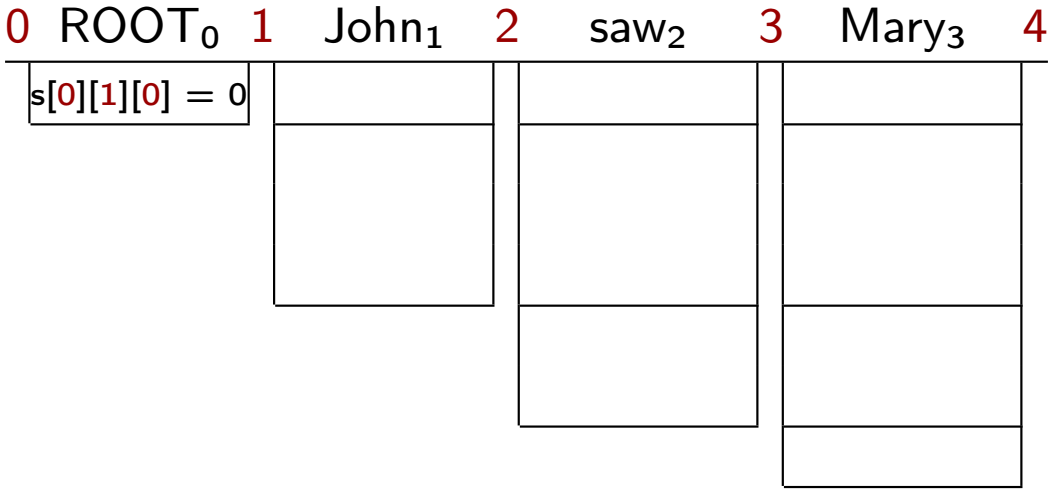
Parsing Algorithm

ROOT_0	$\rightarrow$	John_1	9	saw_2	$\rightarrow$	Mary_3	30
ROOT_0	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	Mary_3	0
ROOT_0	$\rightarrow$	Mary_3	9	John_1	$\rightarrow$	Mary_3	3
John_1	$\rightarrow$	saw_2	20	John_1	$\leftarrow$	Mary_3	11
John_1	$\leftarrow$	saw_2	30				



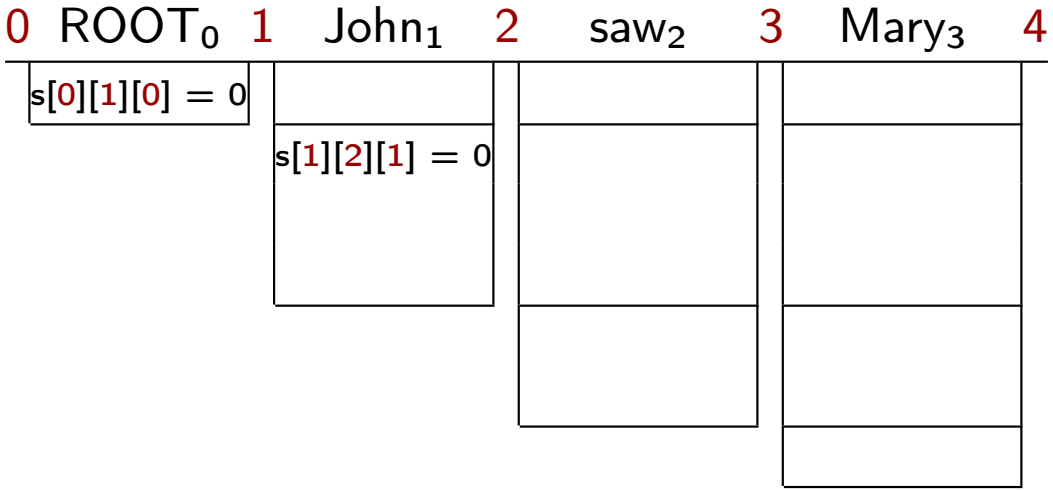
Parsing Algorithm

ROOT_0	$\rightarrow$	John_1	9	saw_2	$\rightarrow$	Mary_3	30
ROOT_0	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	Mary_3	0
ROOT_0	$\rightarrow$	Mary_3	9	John_1	$\rightarrow$	Mary_3	3
John_1	$\rightarrow$	saw_2	20	John_1	$\leftarrow$	Mary_3	11
John_1	$\leftarrow$	saw_2	30				



Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				



Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0	$ROOT_0$	1	$John_1$	2	saw_2	3	$Mary_3$	4
	$s[0][1][0] = 0$							
			$s[1][2][1] = 0$					
					$s[2][3][2] = 0$			

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
$s[0][1][0] = 0$			
	$s[1][2][1] = 0$		
		$s[2][3][2] = 0$	
			$s[3][4][3] = 0$

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0	$ROOT_0$	1	$John_1$	2	saw_2	3	$Mary_3$	4
	$s[0][1][0] = 0$							
			$s[1][2][1] = 0$					
					$s[2][3][2] = 0$			
							$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$			
		$s[2][3][2] = 0$		
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$			
		$s[2][3][2] = 0$		
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$	$s[1][3][1] = 20$		
		$s[2][3][2] = 0$		
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$	$s[1][3][1] = 20$		
		$s[2][3][2] = 0$		
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
$s[0][1][0] = 0$	$s[0][2][0] = 9$		
	$s[1][2][1] = 0$	$s[1][3][1] = 20$ $s[1][3][2] = 30$	
		$s[2][3][2] = 0$	
			$s[3][4][3] = 0$

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$	$s[1][3][1] = 20$ $s[1][3][2] = 30$		
		$s[2][3][2] = 0$		
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9			
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30		
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
$s[0][1][0] = 0$	$s[0][2][0] = 9$		
	$s[1][2][1] = 0$	$s[1][3][1] = 20$ $s[1][3][2] = 30$	
		$s[2][3][2] = 0$	$s[2][4][2] = 30$
			$s[3][4][3] = 0$

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
$s[0][1][0] = 0$	$s[0][2][0] = 9$		
	$s[1][2][1] = 0$	$s[1][3][1] = 20$ $s[1][3][2] = 30$	
		$s[2][3][2] = 0$	$s[2][4][2] = 30$ $s[2][4][3] = 0$
			$s[3][4][3] = 0$

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$			
	$s[1][2][1] = 0$	$s[1][3][1] = 20$ $s[1][3][2] = 30$		
		$s[2][3][2] = 0$	$s[2][4][2] = 30$ $s[2][4][3] = 0$	
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 29		
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30		
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 29		
	s[1][2][1] = 0	s[1][3][1] = 20		
		s[1][3][2] = 30		
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30	
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40		
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30		
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30	
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
$s[0][1][0] = 0$	$s[0][2][0] = 9$	$s[0][3][0] = 40$	
	$s[1][2][1] = 0$	$s[1][3][1] = 20$	
		$s[1][3][2] = 30$	
		$s[2][3][2] = 0$	$s[2][4][2] = 30$
			$s[2][4][3] = 0$
			$s[3][4][3] = 0$

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40		
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30	s[1][4][1] = 50	
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20 s[1][3][2] = 30	s[1][4][1] = 50
		s[2][3][2] = 0	s[2][4][2] = 30 s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40		
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50	
		s[1][3][2] = 30	s[1][4][2] = 60	
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40		
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50	
		s[1][3][2] = 30	s[1][4][2] = 60	
			s[1][4][3] = 11	
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
$s[0][1][0] = 0$	$s[0][2][0] = 9$	$s[0][3][0] = 40$		
	$s[1][2][1] = 0$	$s[1][3][1] = 20$	$s[1][4][1] = 50$	
		$s[1][3][2] = 30$	$s[1][4][2] = 60$	
			$s[1][4][3] = 11$	
		$s[2][3][2] = 0$	$s[2][4][2] = 30$	
			$s[2][4][3] = 0$	
			$s[3][4][3] = 0$	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 59
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 59
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70	
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50	
		s[1][3][2] = 30	s[1][4][2] = 60	
			s[1][4][3] = 11	
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Parsing Algorithm

$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$	4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70	
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50	
		s[1][3][2] = 30	s[1][4][2] = 60	
			s[1][4][3] = 11	
		s[2][3][2] = 0	s[2][4][2] = 30	
			s[2][4][3] = 0	
			s[3][4][3] = 0	

Parsing Algorithm

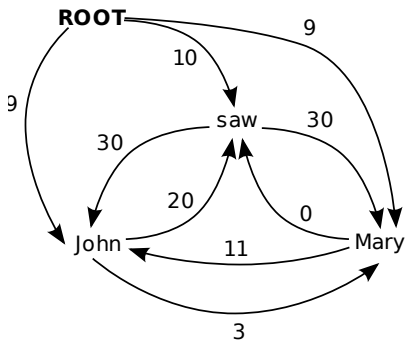
$ROOT_0$	$\rightarrow$	$John_1$	9	saw_2	$\rightarrow$	$Mary_3$	30
$ROOT_0$	$\rightarrow$	saw_2	10	saw_2	$\leftarrow$	$Mary_3$	0
$ROOT_0$	$\rightarrow$	$Mary_3$	9	$John_1$	$\rightarrow$	$Mary_3$	3
$John_1$	$\rightarrow$	saw_2	20	$John_1$	$\leftarrow$	$Mary_3$	11
$John_1$	$\leftarrow$	saw_2	30				

0 $ROOT_0$	1 $John_1$	2 saw_2	3 $Mary_3$ 4
s[0][1][0] = 0	s[0][2][0] = 9	s[0][3][0] = 40	s[0][4][0] = 70
	s[1][2][1] = 0	s[1][3][1] = 20	s[1][4][1] = 50
		s[1][3][2] = 30	s[1][4][2] = 60
			s[1][4][3] = 11
		s[2][3][2] = 0	s[2][4][2] = 30
			s[2][4][3] = 0
			s[3][4][3] = 0

Non-Projective Parsing

- Dynamic programming only works for projective trees
- Arc-factored non-projective parsing:
  - Maximum spanning tree algorithm from graph theory
  - Starts from greedy solution (best head for each word)
  - If this is a tree, it must be the best tree
  - If not, contract a cycle and apply algorithm recursively
- This does not work for more complex models

Quiz



0	ROOT_0	1	John_1	2	saw_2	3	Mary_3	4

- What is the value of $S[1][3][1]$ after filling the parse table?
  1. 0
  2. 20
  3. 30
  4. 50

Scores and Machine Learning

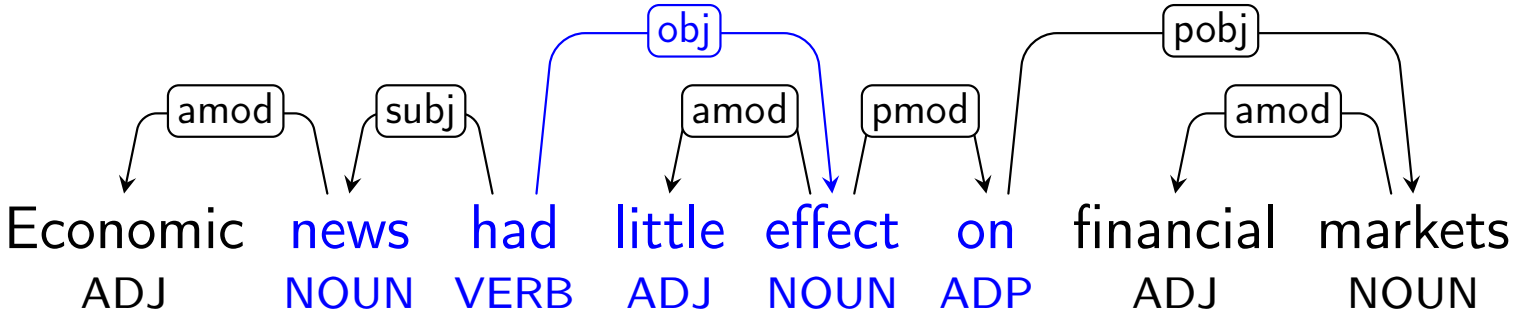
- We can use machine learning to score subgraphs (arcs)
- Given a treebank:
  - We know the dependency tree for each sentence
  - We can factor the tree into its subgraphs
  - We can learn to give high scores to subgraphs of good trees
- We represent each subgraph by a finite set of features

Features

- Features are restricted (because of parsing algorithms)
  - Features of subgraph (single arc)
  - Features of input sentence

Features

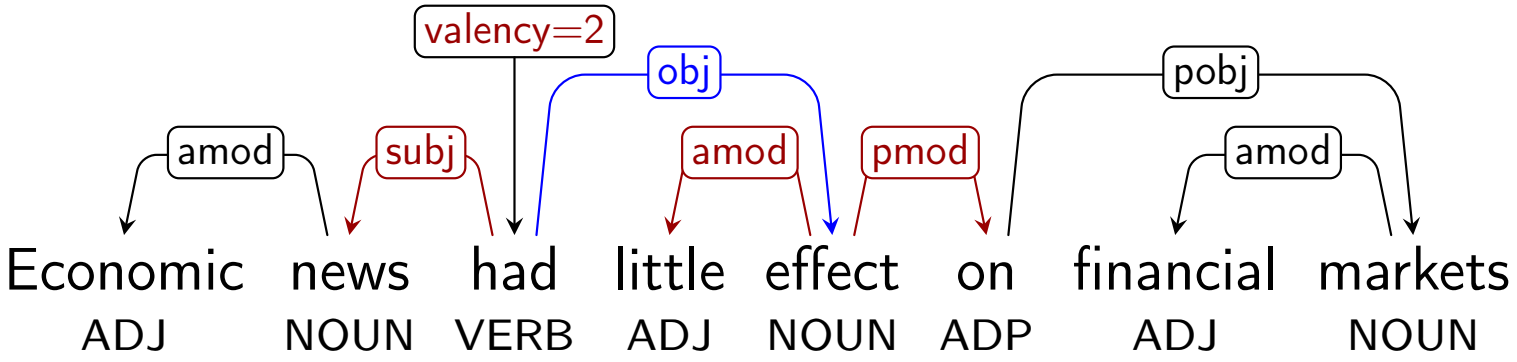
- Features are restricted (because of parsing algorithms)
  - Features of subgraph (single arc)
  - Features of input sentence
- For arc (had, effect) below, we can have features over properties of arc and context within sentence



Features

- Features are restricted (because of parsing algorithms)
  - Features of subgraph (single arc)
  - Features of input sentence

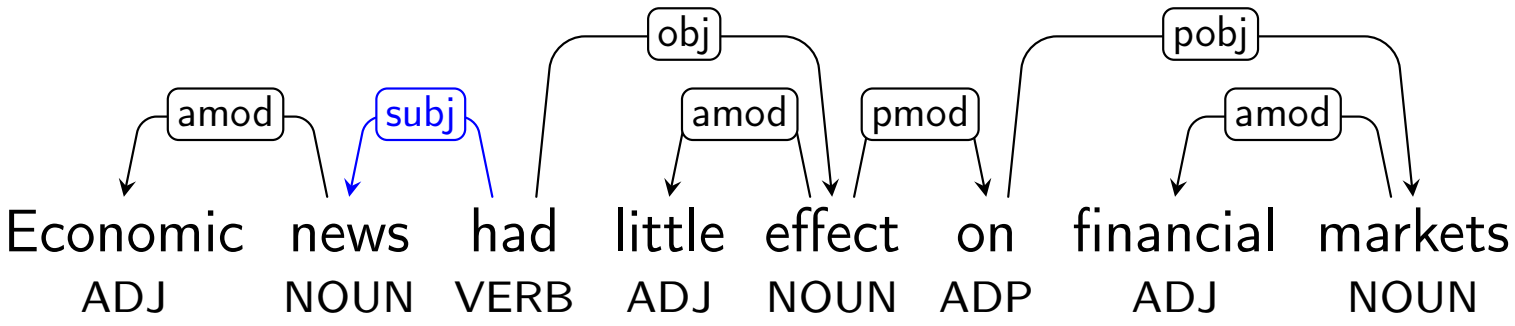
- For arc (had, effect) below, we cannot have features over multiple arcs (siblings, grandparents), valency, etc.



Graph-Based Parsing

- Advantage:
  - Exact search for best tree – no search errors
- Drawbacks:
  - Feature scope limited – especially for arc-factored model
  - Less efficient than transition-based parsing (cubic vs. linear)

Quiz



- Which features can be used to score the arc (had, news)?
  1. pos(had) = VERB
  2. pos(little) = ADJ
  3. head(news) = had
  4. head(little) = effect