Dependency Parsing

Gabriele A. Musillo
Fall 2014

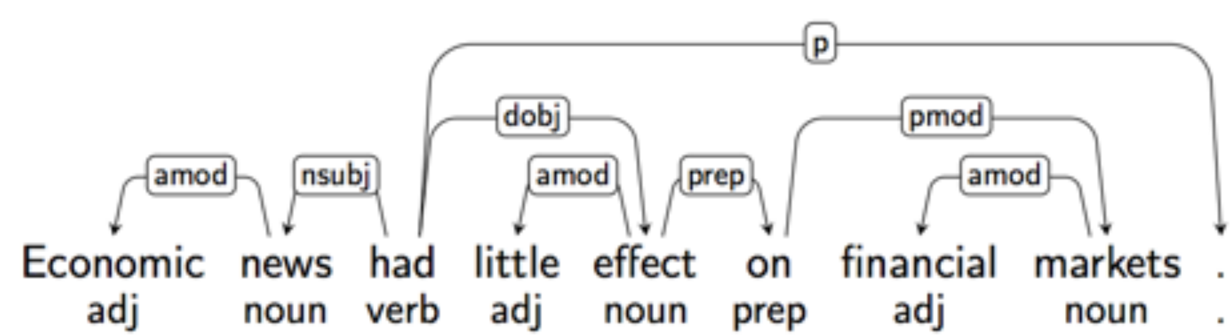
Thanks to Joakim Nivre and Ryan McDonald for sharing their slides.

Overview

- Dependency Syntax
- Transition-based Parsing

Configuration:	(S, B, A)		
Initial:	$([\ ], [0, 1, \ldots, n], \{\ \})$		
Terminal:	$(S, [\ ], A)$		
Shift:	$(S, i \vert B, A)$	$\Rightarrow$	$(S \vert i, B, A)$
Reduce:	$(S \vert i, B, A)$	$\Rightarrow$	(S, B, A)
Right-Arc(k):	$(S \vert i, j \vert B, A)$	$\Rightarrow$	$(S \vert i \vert j, B, A \cup \{(i, j, k)\})$
Left-Arc(k):	$(S \vert i, j \vert B, A)$	$\Rightarrow$	$(S, j \vert B, A \cup \{(j, i, k)\})$

$\Leftrightarrow$



Dependency Syntax

- The basic idea:
  - Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.
- In the words of Lucien Tesnière [Tesnière 1959]:
  - La phrase est un *ensemble organisé* dont les éléments constituants sont les *mots*. [1.2] Tout mot qui fait partie d'une phrase cesse par lui-même d'être isolé comme dans le dictionnaire. Entre lui et ses voisins, l'esprit aperçoit des *connexions*, dont l'ensemble forme la charpente de la phrase. [1.3] Les connexions structurales établissent entre les mots des rapports de *dépendance*. Chaque connexion unit en principe un terme *supérieur* à un terme *inférieur*. [2.1] Le terme supérieur reçoit le nom de *régissant*. Le terme inférieur reçoit le nom de *subordonné*. Ainsi dans la phrase *Alfred parle* [. . .], *parle* est le régissant et *Alfred* le subordonné. [2.2]

Syntactic Dependencies

Economic	news	had	little	effect	on	financial	markets	.
adj	noun	verb	adj	noun	prep	adj	noun	.

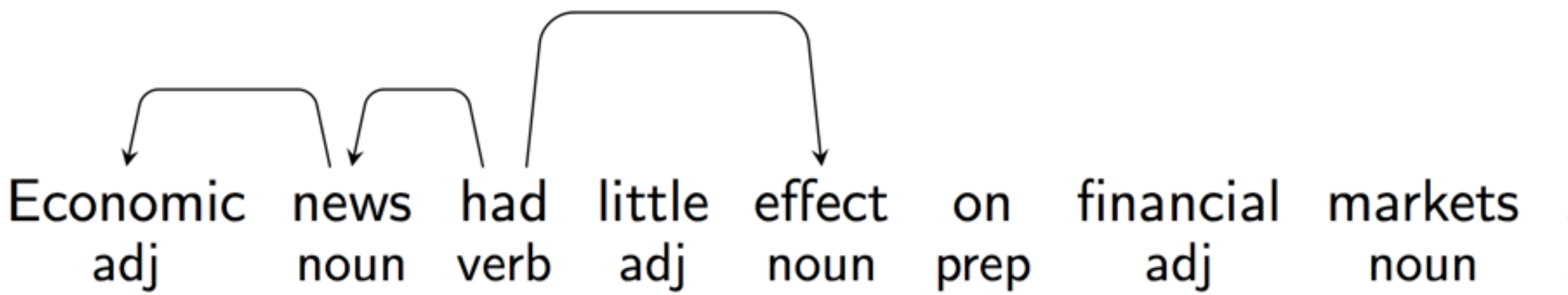
Syntactic Dependencies

Economic	news	had	little	effect	on	financial	markets	.
adj	noun	verb	adj	noun	prep	adj	noun	.

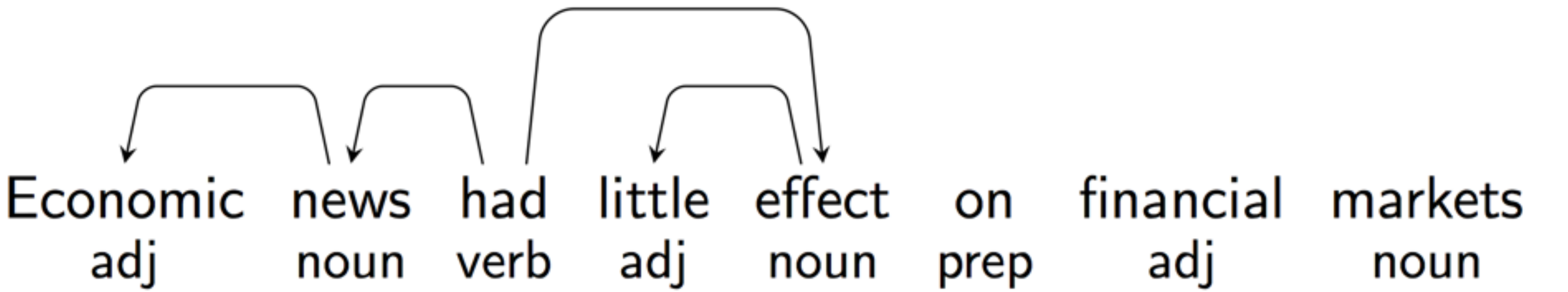
Syntactic Dependencies

Economic	news	had	little	effect	on	financial	markets	.
adj	noun	verb	adj	noun	prep	adj	noun	.

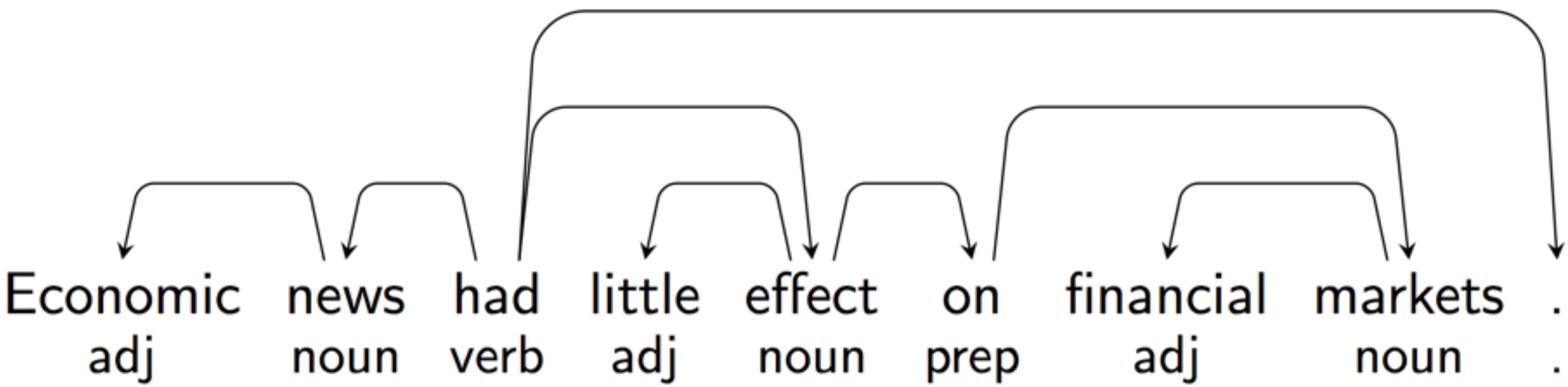
Syntactic Dependencies



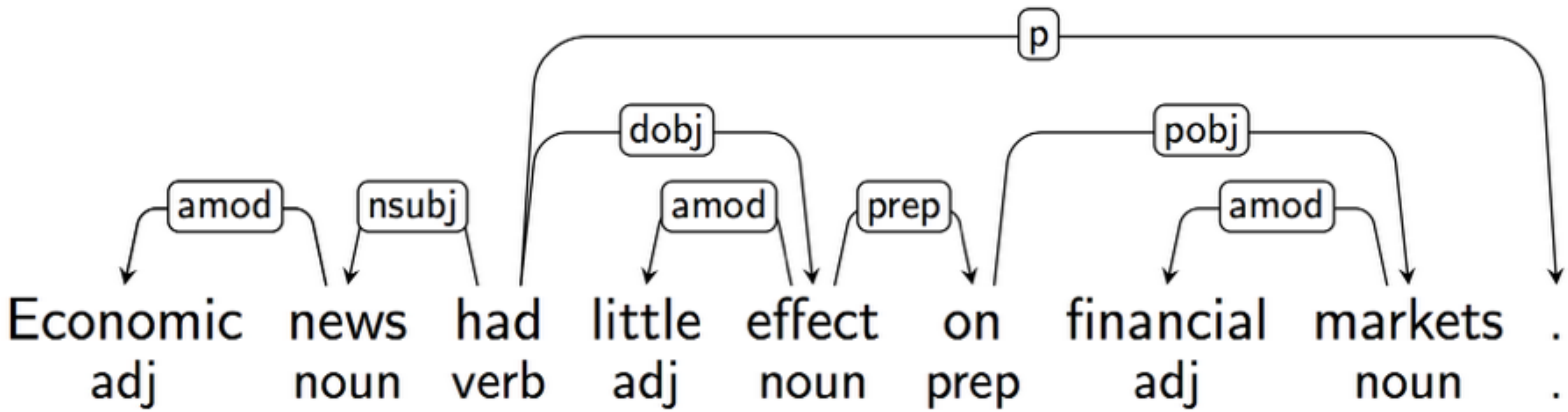
Syntactic Dependencies



Syntactic Dependencies

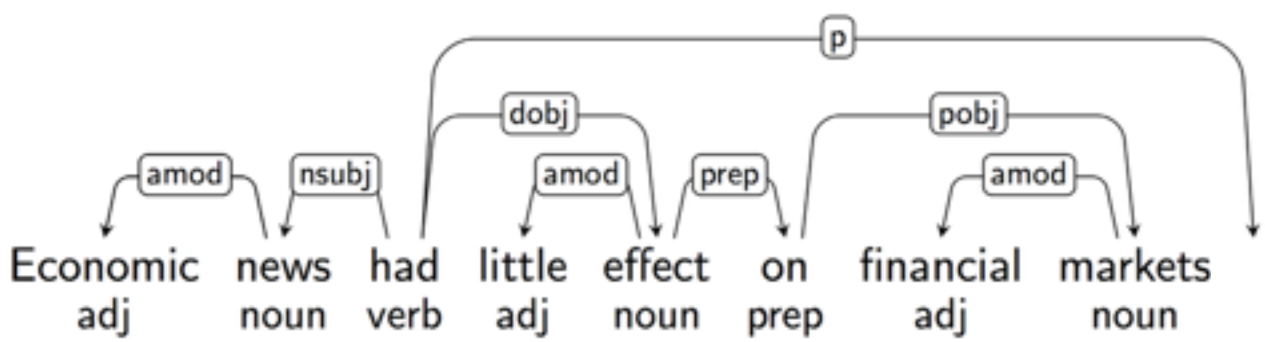


Syntactic Dependencies

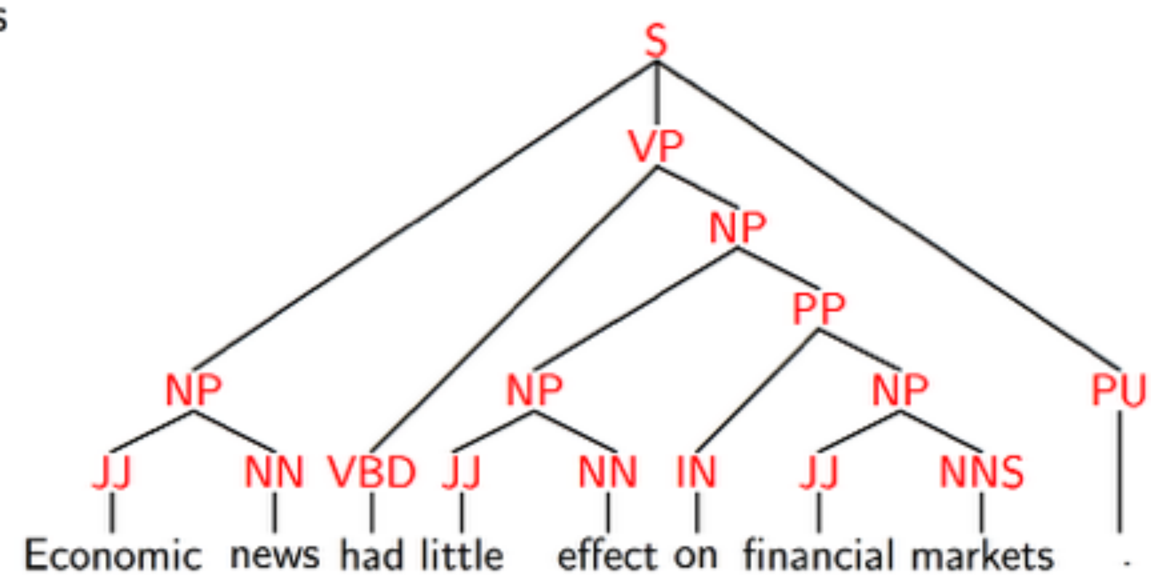


Dependency vs. Constituency Syntax

- head-dependent relations (directed arcs),
- functional categories (arc labels),
- possibly some structural categories (parts-of-speech)



- phrases (nonterminal nodes),
- structural categories (nonterminal labels),
- possibly some functional categories (grammatical functions)



Dependency Graphs

- A dependency structure can be defined as a directed graph G, consisting of
  - a set V of nodes (vertices),
  - a set A of arcs (directed edges),
  - a linear precedence order $<$ on V (word order).
- Labeled graphs:
  - Nodes in V are labeled with word forms (and annotation).
  - Arcs in A are labeled with dependency types:
    - $L = \{l_1, \ldots, l_{|L|}\}$ is the set of permissible arc labels.
    - Every arc in A is a triple (i, j, k), representing a dependency from w_i to w_j with label l_k.

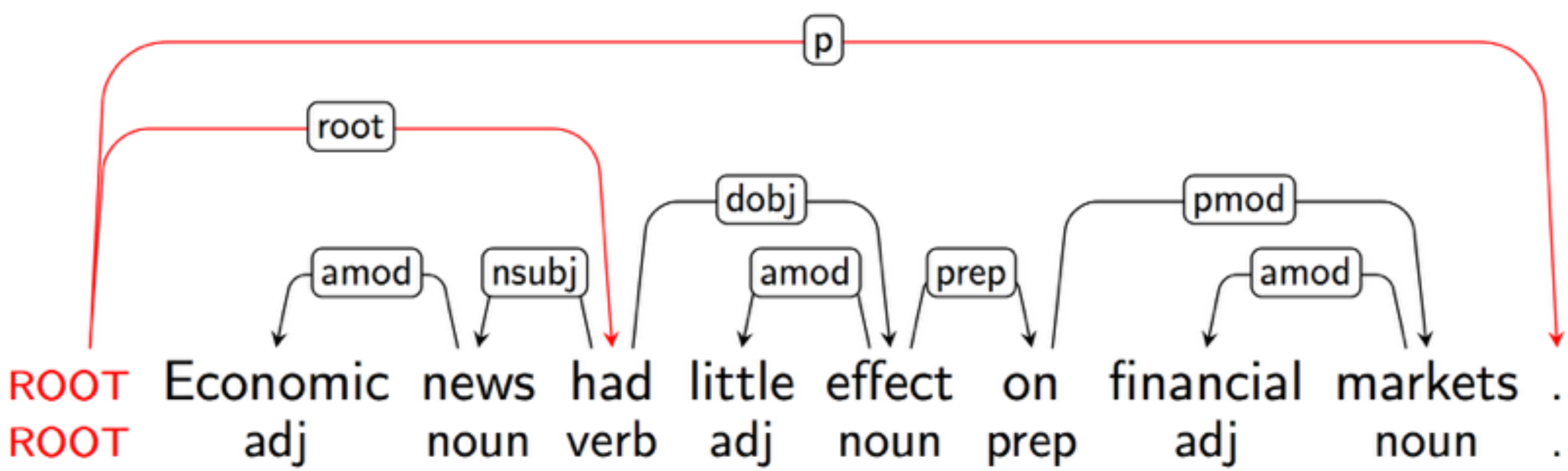
Formal Properties of Dependency Graphs

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$
  - $i \to j \;\equiv\; \exists k : (i, j, k) \in A$
  - $i \leftrightarrow j \;\equiv\; i \to j \lor j \to i$
  - $i \to^* j \;\equiv\; i = j \lor \exists i' : i \to i', i' \to^* j$
  - $i \leftrightarrow^* j \;\equiv\; i = j \lor \exists i' : i \leftrightarrow i', i' \leftrightarrow^* j$

- G is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.
- G is acyclic:
  - If $i \to j$, then not $j \to^* i$.
- G obeys the single-head constraint:
  - If $i \to j$, then not $i' \to j$, for any $i' \neq i$.
- G is projective:
  - If $i \to j$, then $i \to^* i'$, for any i' such that $i < i' < j$ or $j < i' < i$.

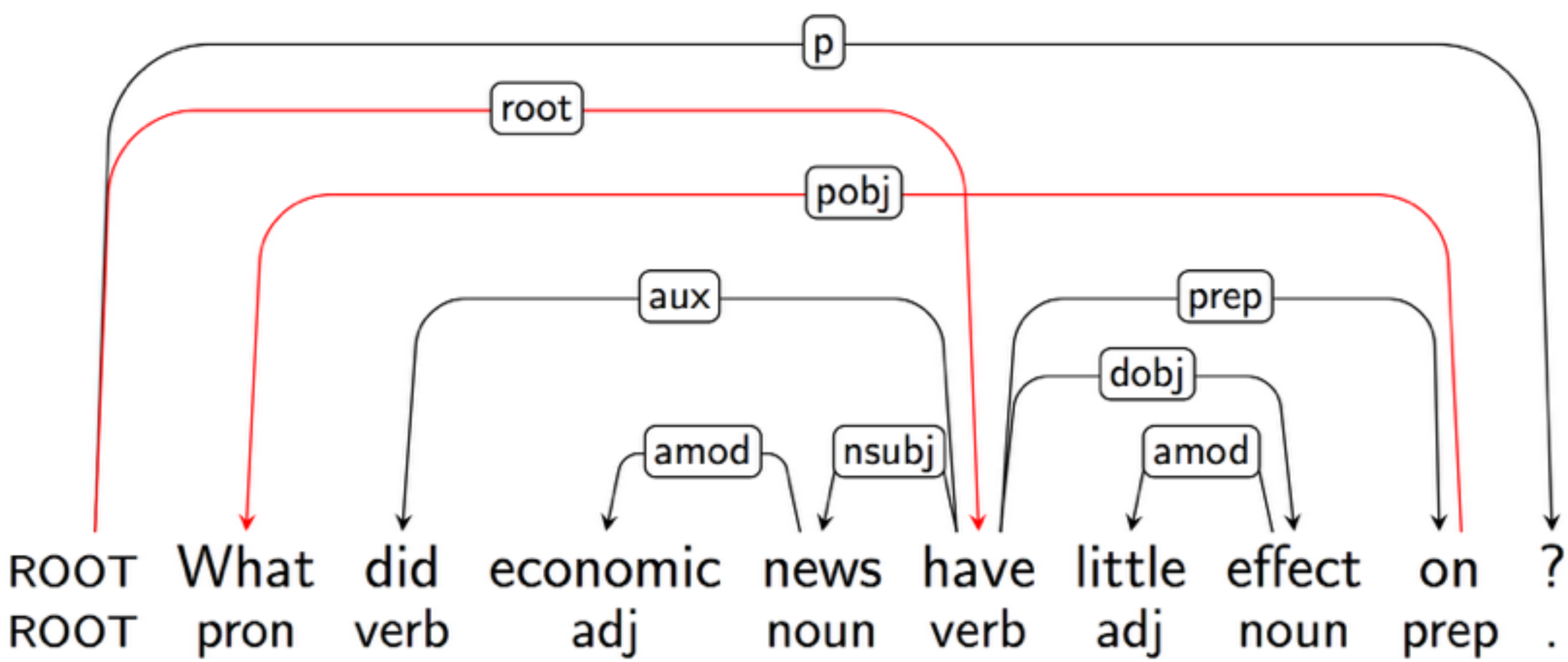
Projective Dependency Trees

- Single-Headedness
- Acyclicity
- Connectedness (by adding a special root node)
- Projective



Non-Projective Dependency Trees

- Long-Distance Dependencies
- Free Word Order



Dependency Parsing

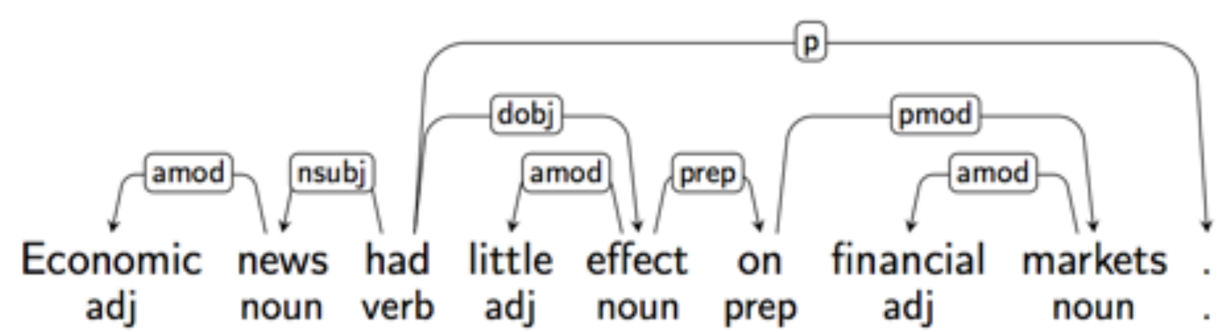
- The problem:
  - Input: Sentence $x = w_0, w_1, \ldots, w_n$ with $w_0 = \text{ROOT}$
  - Output: Dependency graph $G = (V, A)$ for x where:
    - $V = \{0, 1, \ldots, n\}$ is the vertex set,
    - A is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from w_i to w_j with label $l_k \in L$
- Two main approaches:
  - Grammar-based parsing
    - Context-free dependency grammar
    - Lexicalized context-free grammars
    - Constraint dependency grammar
  - Data-driven parsing
    - Graph-based models
    - Transition-based models
    - Easy-first parsing
    - Hybrids: grammar+data-driven, ensembles, etc.

Transition-Based Models

- Define a transition system for dependency parsing
- Learn a model for scoring possible transitions
- Parse by searching for the optimal transition sequence

Configuration:	(S, B, A)		
Initial:	$([\,], [0, 1, \ldots, n], \{\,\})$		
Terminal:	$(S, [\,], A)$		
Shift:	$(S, i \mid B, A)$	$\Rightarrow$	$(S \mid i, B, A)$
Reduce:	$(S \mid i, B, A)$	$\Rightarrow$	(S, B, A)
Right-Arc(k):	$(S \mid i, j \mid B, A)$	$\Rightarrow$	$(S \mid i \mid j, B, A \cup \{(i, j, k)\})$
Left-Arc(k):	$(S \mid i, j \mid B, A)$	$\Rightarrow$	$(S, j \mid B, A \cup \{(j, i, k)\})$

$\Leftrightarrow$



Arc-Eager Transition System

Configuration:	(S, B, A)	$[S = \text{Stack},\ B = \text{Buffer},\ A = \text{Arcs}]$		
Initial:	$([\ ], [0, 1, \ldots, n], \{\ \})$			
Terminal:	$(S, [\ ], A)$			
Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$	
Reduce:	$(S\|i, B, A)$	$\Rightarrow$	(S, B, A)	$h(i, A)$
Right-Arc(k):	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S\|i\|j, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S, j\|B, A \cup \{(j, i, k)\})$	$\neg h(i, A) \wedge i \neq 0$

Notation: $S|i$ = stack with top i and remainder S
$j|B$ = buffer with head j and remainder B
$h(i, A)$ = i has a head in A

Shift

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

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

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

Shift

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

↓

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

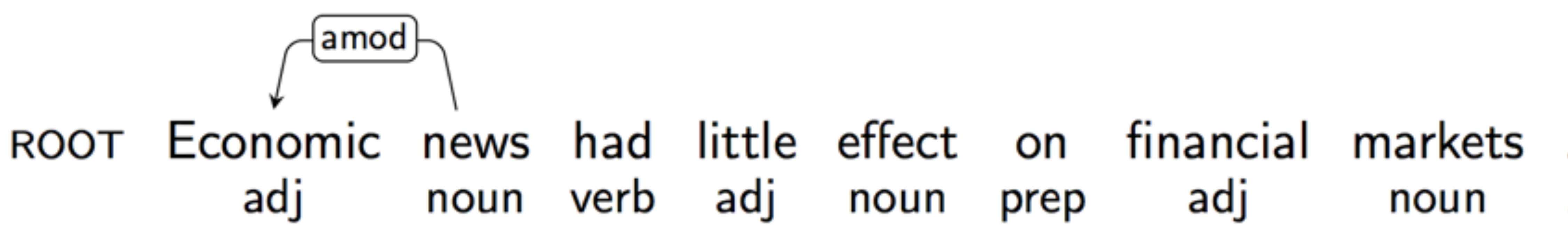
ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

Left-Arc(a*mod*)

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

↓

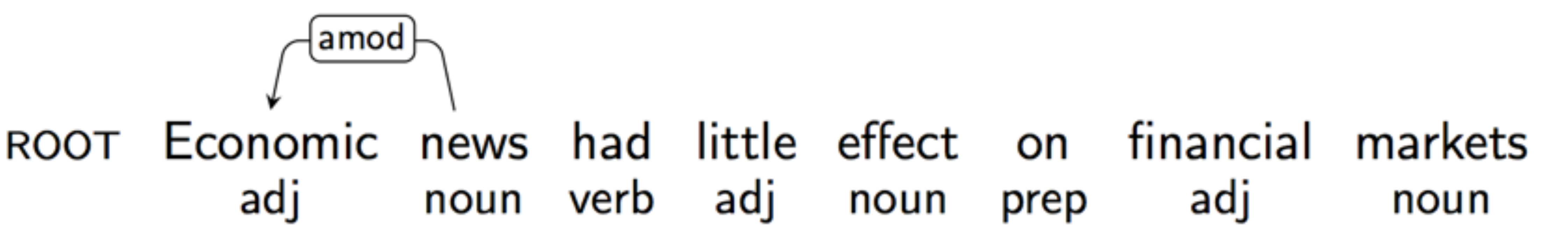
[ROOT]   [news, had, little, effect, on, financial, markets, .]



Shift

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

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

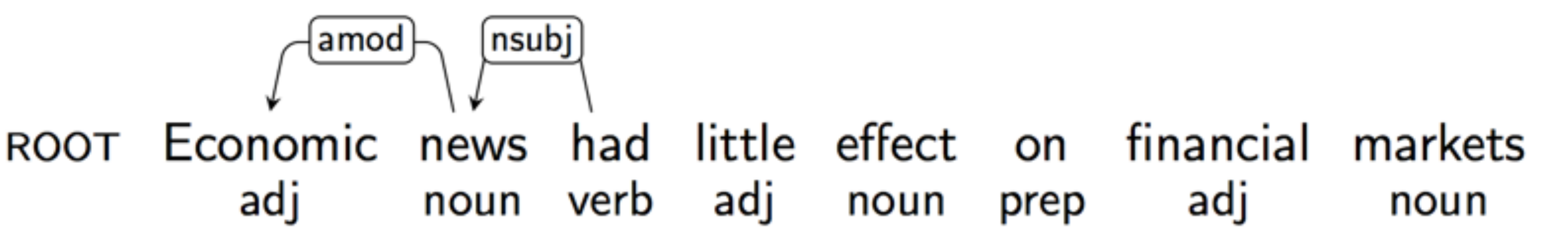


Left-Arc(*nsubj*)

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

↓

[ROOT]   [had, little, effect, on, financial, markets, .]

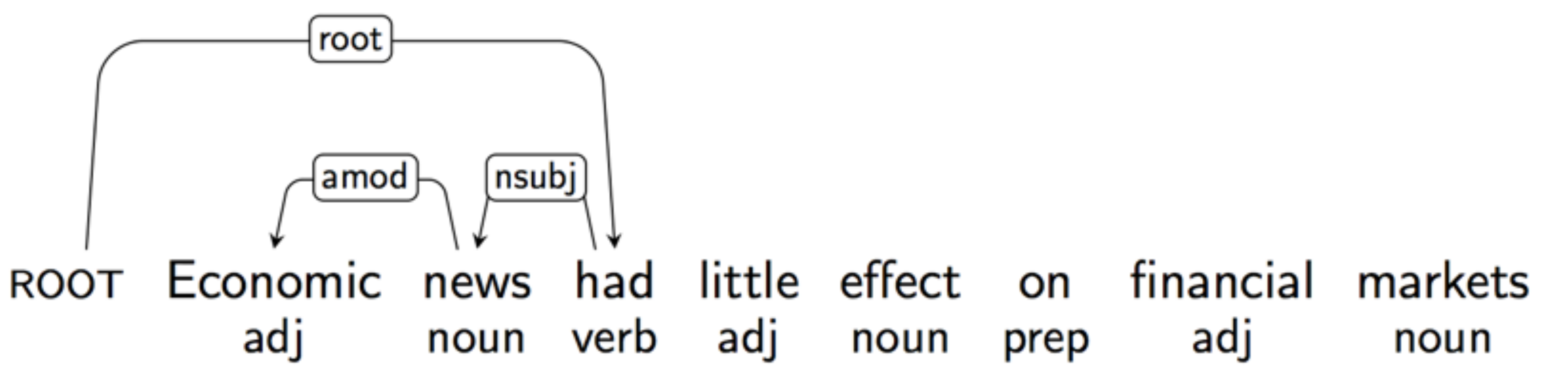


Right-Arc(*root*)

[ROOT] [had, little, effect, on, financial, markets, .]

↓

[ROOT, had] [little, effect, on, financial, markets, .]

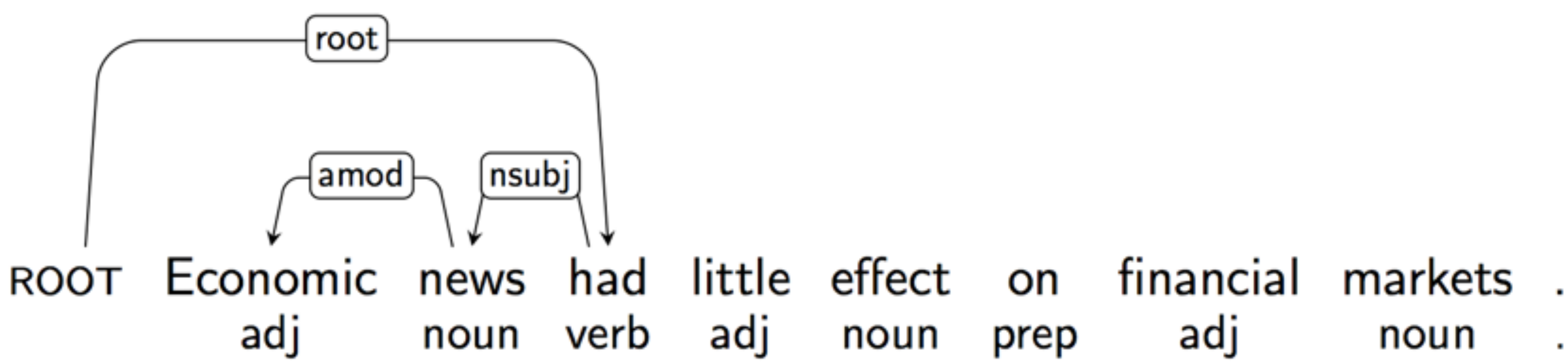


Shift

[ROOT, had] [little, effect, on, financial, markets, .]

↓

[ROOT, had, little] [effect, on, financial, markets, .]

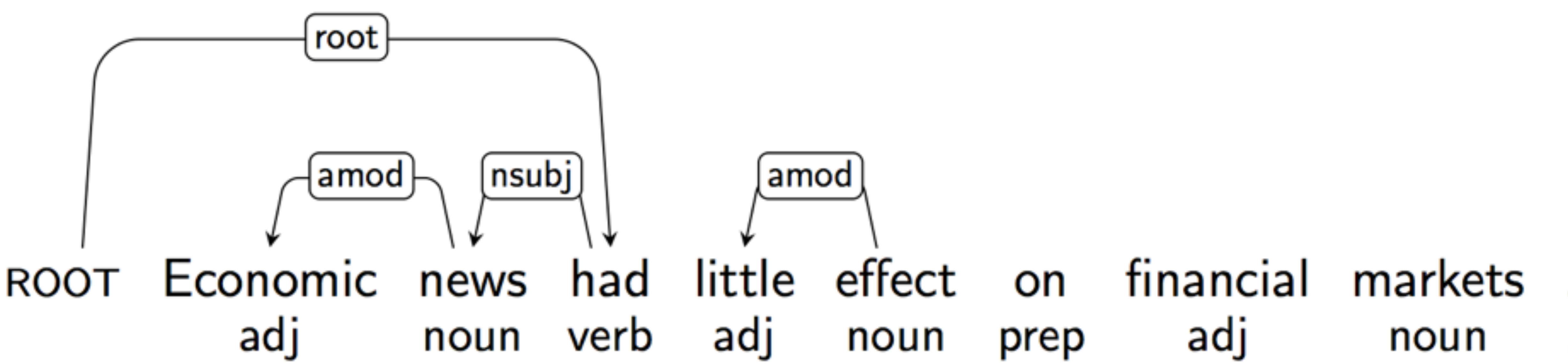


Left-Arc(*amod*)

[ROOT, had, little] [effect, on, financial, markets, .]

↓

[ROOT, had] [effect, on, financial, markets, .]

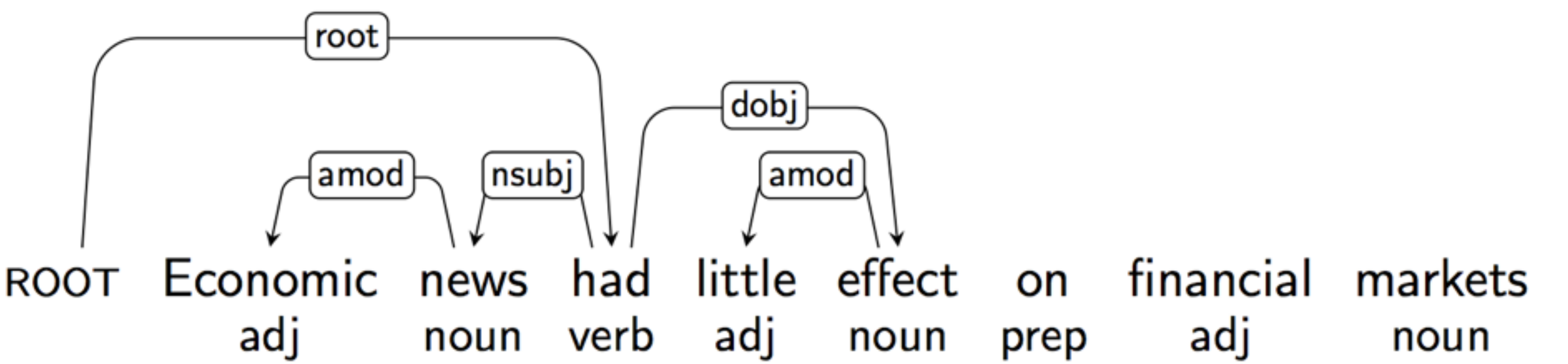


Right-Arc(*dobj*)

[ROOT, had] [effect, on, financial, markets, .]

↓

[ROOT, had, effect] [on, financial, markets, .]

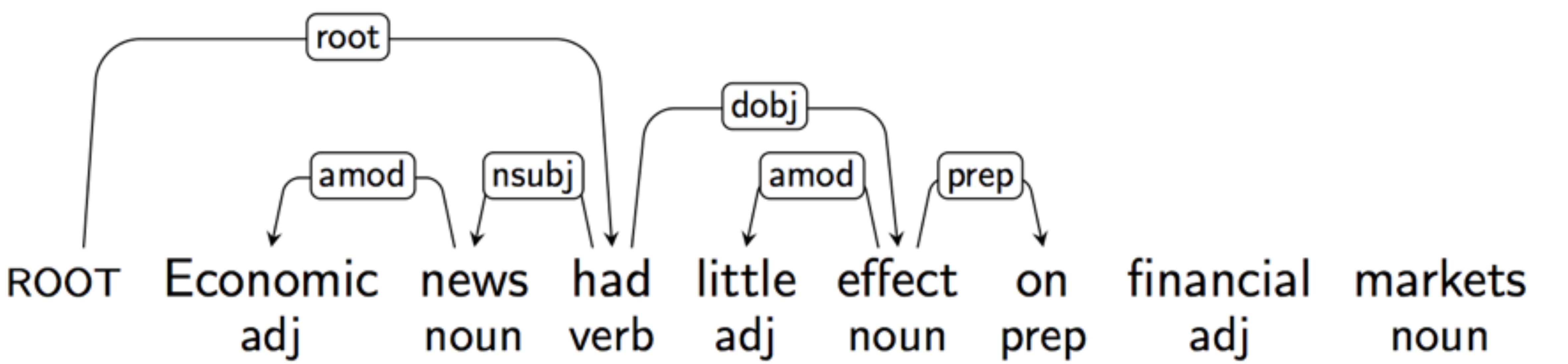


Right-Arc(prep)

[ROOT, had, effect]   [on, financial, markets, .]

↓

[ROOT, had, effect, on]   [financial, markets, .]

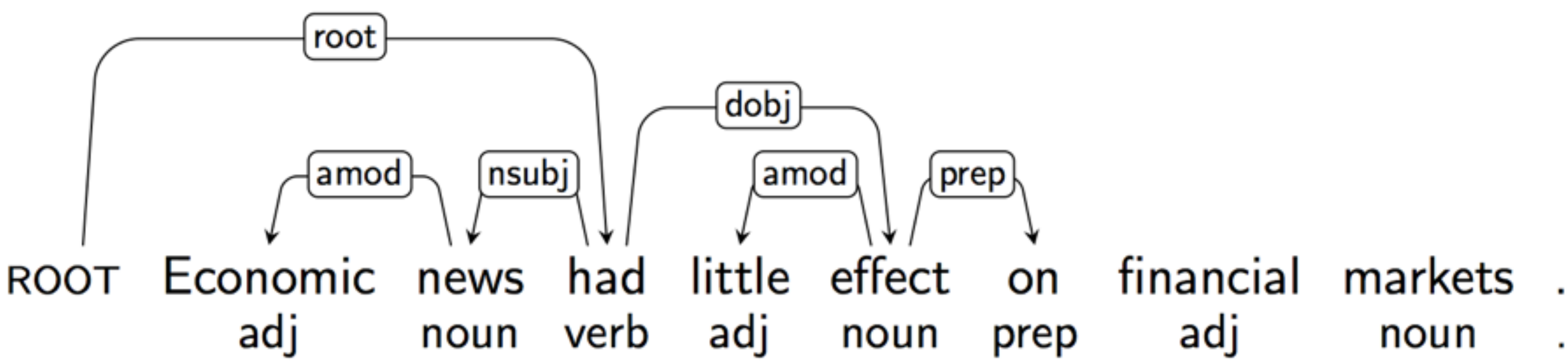


Shift

[ROOT, had, effect, on]   [financial, markets, .]

↓

[ROOT, had, effect, on, financial]   [markets, .]

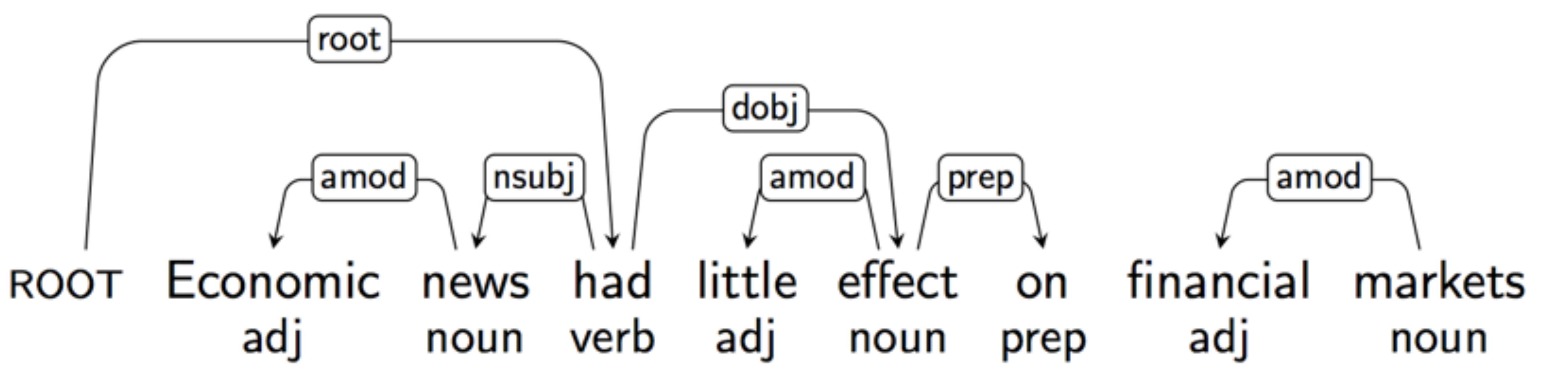


Left-Arc(*amod*)

[ROOT, had, effect, on, financial] [markets, .]

↓

[ROOT, had, effect, on] [markets, .]

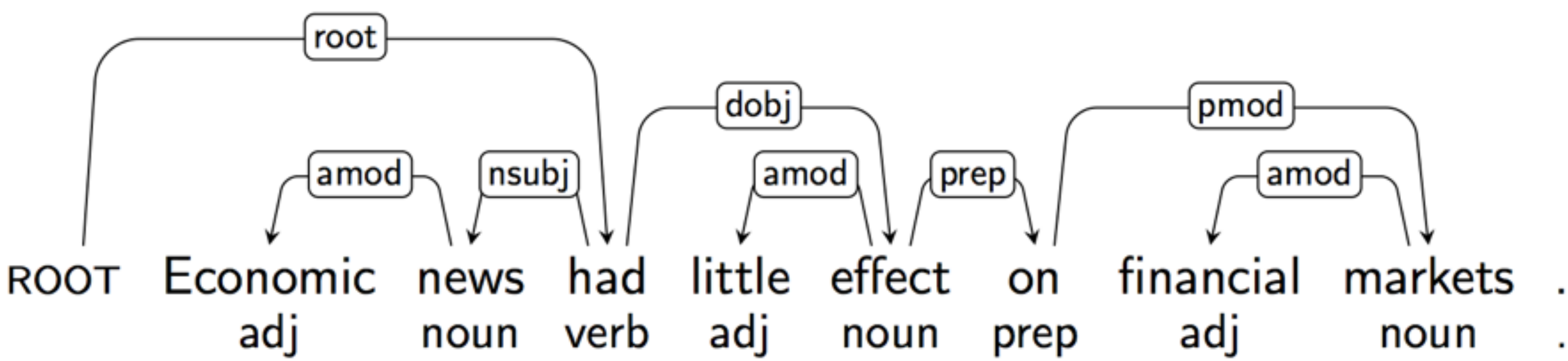


Right-Arc(*pmod*)

[ROOT, had, effect, on] [markets, .]

↓

[ROOT, had, effect, on, markets] [.]

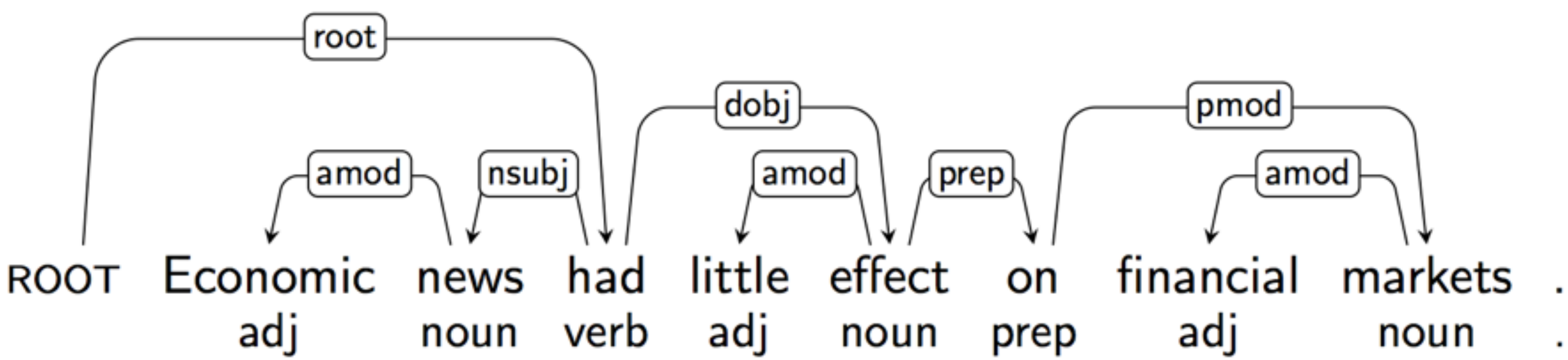


Reduce

[ROOT, had, effect, on, markets]   [.]

↓

[ROOT, had, effect, on]   [.]

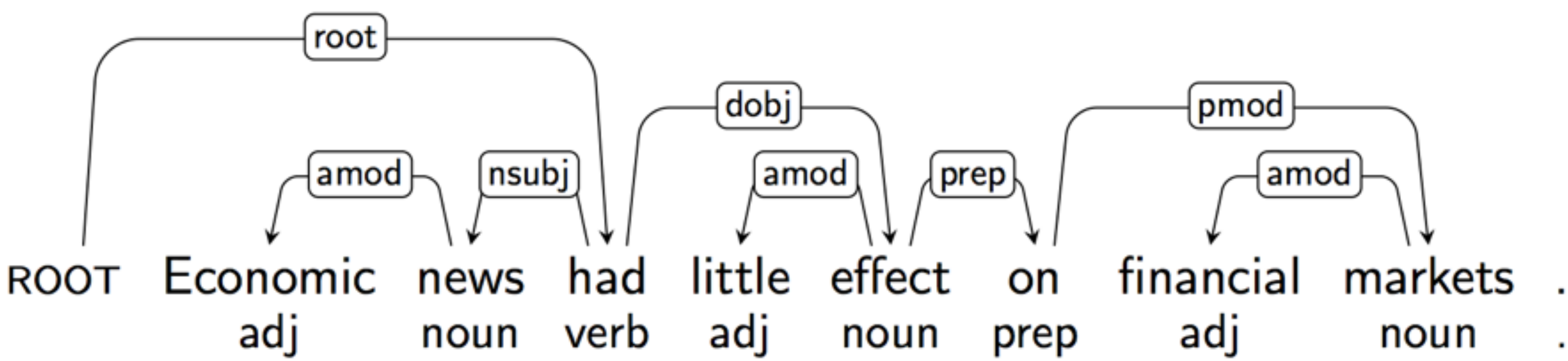


Reduce

[ROOT, had, effect, on] [.]

↓

[ROOT, had, effect] [.]

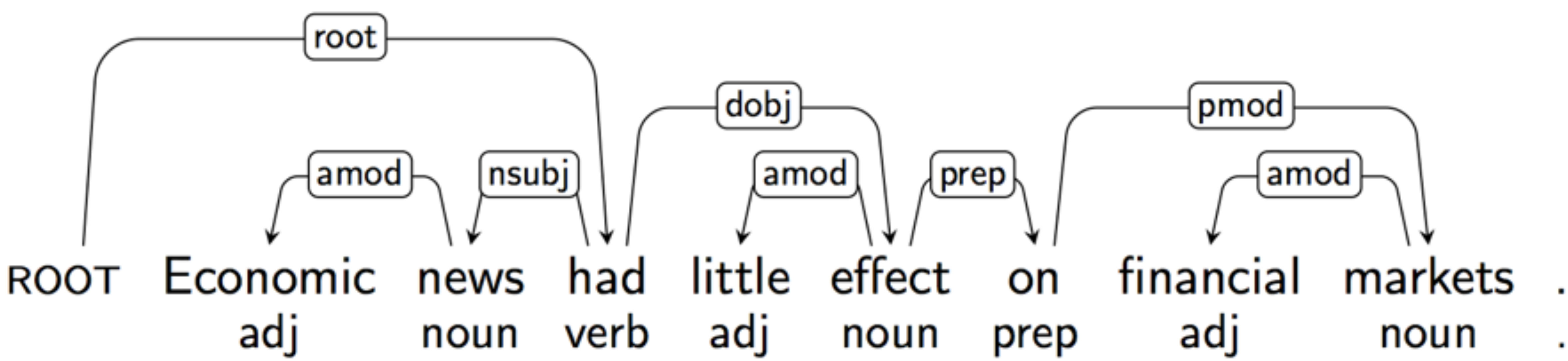


Reduce

[ROOT, had, effect] [.]

↓

[ROOT, had] [.]

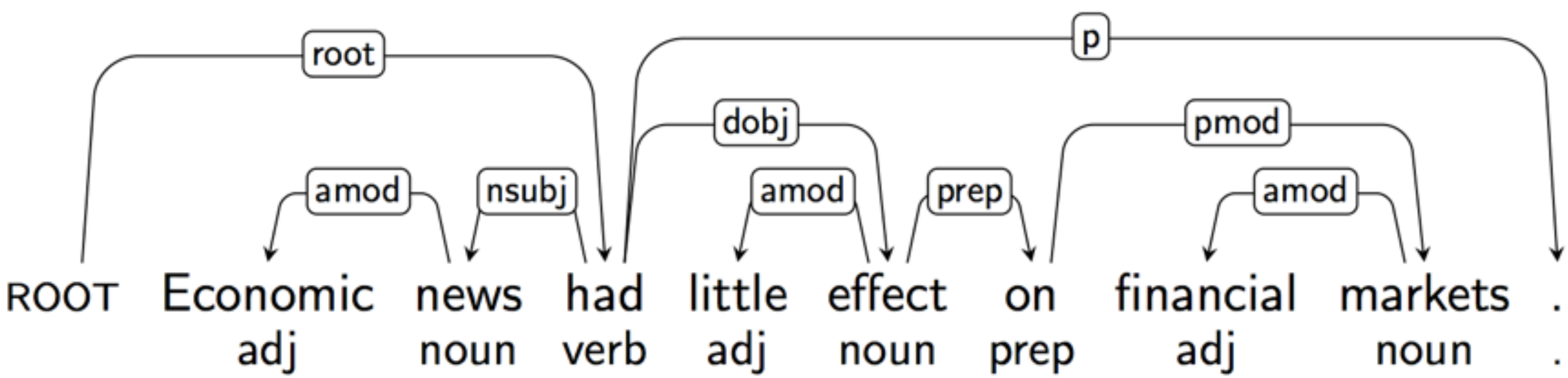


Right-Arc(p)

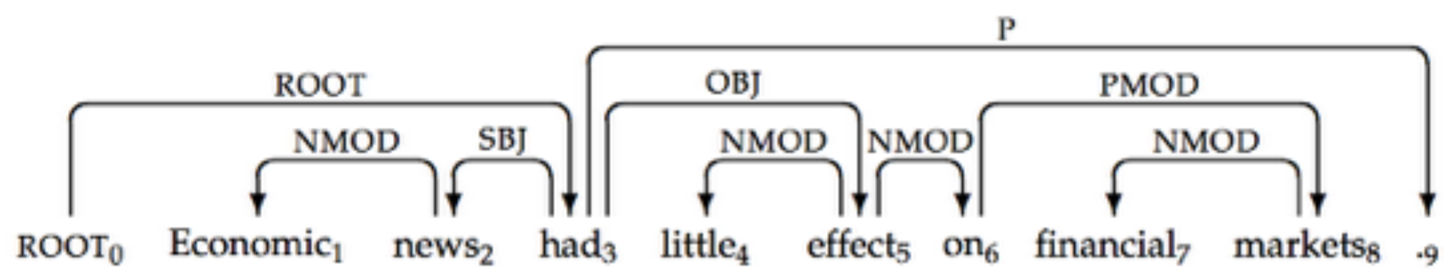
[ROOT, had]   [.]

↓

[ROOT, had, .]   []



Arc-Eager Transition Sequence



Transition	Configuration				
	(	[0],	[1, . . . , 9],	$\emptyset$	)
SHIFT $\Longrightarrow$	(	[0, 1],	[2, . . . , 9],	$\emptyset$	)
LEFT-ARC$_{\text{NMOD}}$ $\Longrightarrow$	(	[0],	[2, . . . , 9],	$A_1 = \{(2, \text{NMOD}, 1)\}$	)
SHIFT $\Longrightarrow$	(	[0, 2],	[3, . . . , 9],	A_1	)
LEFT-ARC$_{\text{SBJ}}$ $\Longrightarrow$	(	[0],	[3, . . . , 9],	$A_2 = A_1 \cup \{(3, \text{SBJ}, 2)\}$	)
RIGHT-ARC$^e_{\text{ROOT}}$ $\Longrightarrow$	(	[0, 3],	[4, . . . , 9],	$A_3 = A_2 \cup \{(0, \text{ROOT}, 3)\}$	)
SHIFT $\Longrightarrow$	(	[0, 3, 4],	[5, . . . , 9],	A_3	)
LEFT-ARC$_{\text{NMOD}}$ $\Longrightarrow$	(	[0, 3],	[5, . . . , 9],	$A_4 = A_3 \cup \{(5, \text{NMOD}, 4)\}$	)
RIGHT-ARC$^e_{\text{OBJ}}$ $\Longrightarrow$	(	[0, 3, 5],	[6, . . . , 9],	$A_5 = A_4 \cup \{(3, \text{OBJ}, 5)\}$	)
RIGHT-ARC$^e_{\text{NMOD}}$ $\Longrightarrow$	(	[0, . . . , 6],	[7, 8, 9],	$A_6 = A_5 \cup \{(5, \text{NMOD}, 6)\}$	)
SHIFT $\Longrightarrow$	(	[0, . . . , 7],	[8, 9],	A_6	)
LEFT-ARC$_{\text{NMOD}}$ $\Longrightarrow$	(	[0, . . . 6],	[8, 9],	$A_7 = A_6 \cup \{(8, \text{NMOD}, 7)\}$	)
RIGHT-ARC$^e_{\text{PMOD}}$ $\Longrightarrow$	(	[0, . . . , 8],	[9],	$A_8 = A_7 \cup \{(6, \text{PMOD}, 8)\}$	)
REDUCE $\Longrightarrow$	(	[0, . . . , 6],	[9],	A_8	)
REDUCE $\Longrightarrow$	(	[0, 3, 5],	[9],	A_8	)
REDUCE $\Longrightarrow$	(	[0, 3],	[9],	A_8	)
RIGHT-ARC$^e_{\text{P}}$ $\Longrightarrow$	(	[0, 3, 9],	[ ],	$A_9 = A_8 \cup \{(3, \text{P}, 9)\}$	)

Oracle Transition Sequence

Algorithm 1 ORACLE-PARSER($x = (w_1, \ldots, w_n), h, d$)

$c = c_0$ {initial configuration}
$T = \emptyset$ {initial empty transition sequence}
while $c = (\sigma, \tau, h, d)$ is not terminal **do**
  if $\sigma = \epsilon$ **then**
    $t = SHIFT$
  else
    $t =$ ORACLE(σ, τ, h, d) {oracle transition}
  $T = T + t$ {update transition sequence with transition t}
  $c = t(c)$ {configuration resulting from transition t}
return T

Algorithm 2 ORACLE($c = (\sigma|i, j|\tau, h, d)$)

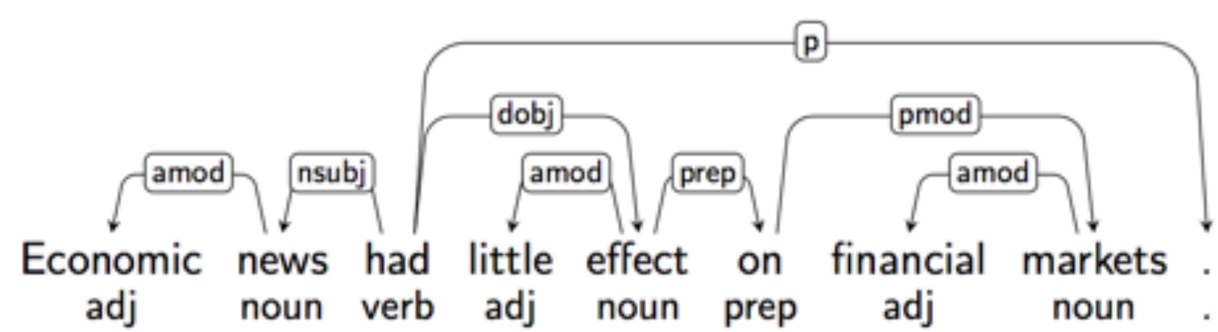
if $h(i) = j$ **then**
  return LEFT-ARC($d(i)$)
else if $h(j) = i$ **then**
  return RIGHT-ARC($d(j)$)
else if $\exists k \in \sigma(h(j) = k \vee h(k) = j)$ **then**
  return REDUCE
else
  return SHIFT

Transition-Based Models

- Define a transition system for dependency parsing
- Learn a model for scoring possible transitions
- Parse by searching for the optimal transition sequence

Configuration:	(S, B, A)		
Initial:	$([\,], [0, 1, \ldots, n], \{\,\})$		
Terminal:	$(S, [\,], A)$		
Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$
Reduce:	$(S\|i, B, A)$	$\Rightarrow$	(S, B, A)
Right-Arc(k):	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S\|i\|j, B, A \cup \{(i, j, k)\})$
Left-Arc(k):	$(S\|i, j\|B, A)$	$\Rightarrow$	$(S, j\|B, A \cup \{(j, i, k)\})$

$\Leftrightarrow$



Learning Oracle Transitions

- An oracle can be approximated by a (linear) classifier:

$$o(c) = \operatorname*{argmax}_{t} \mathbf{w} \cdot \mathbf{f}(c, t)$$

- History-based feature representation $\mathbf{f}(c, t)$
- Weight vector $\mathbf{w}$ learned from treebank data

Learning Oracle Transitions

- Given a treebank:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set $D = \{(c, t) \mid o(c) = t\}$
  - Maximize accuracy of local predictions $o(c) = t$
- Any (unstructured) classifier will do (SVMs are popular)
- Training is local and restricted to oracle configurations

Greedy Parsing

Parse($w_1, \ldots, w_n$)

$c \leftarrow ([\ ]_S, [0, 1, \ldots, n]_B, \{\ \})$

while $B_c \neq [\ ]$

  $t \leftarrow o(c)$

  $c \leftarrow t(c)$

return $G = (\{0, 1, \ldots, n\}, A_c)$

- Complexity given by upper bound on number of transitions
- Parsing in $O(n)$ time for the arc-eager transition system

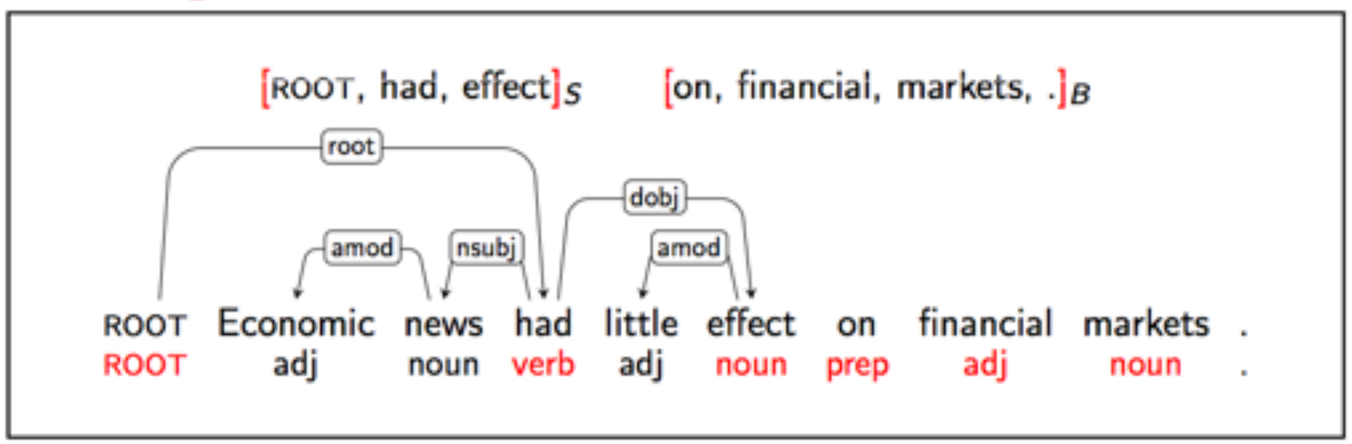
Linear-Time Transition-Based Parsing

- ***o(c)*** and ***t(c)*** can be computed in constant time
- **PUSH** transitions: Shift and Right-Arc
- **POP** transitions: Left-Arc and Reduce
- at most ***n*** PUSH transitions to reach a terminal configuration from the initial one
- POP transitions bounded by PUSH transitions

Static PoS Features

- Features over input tokens relative to S and B

Configuration



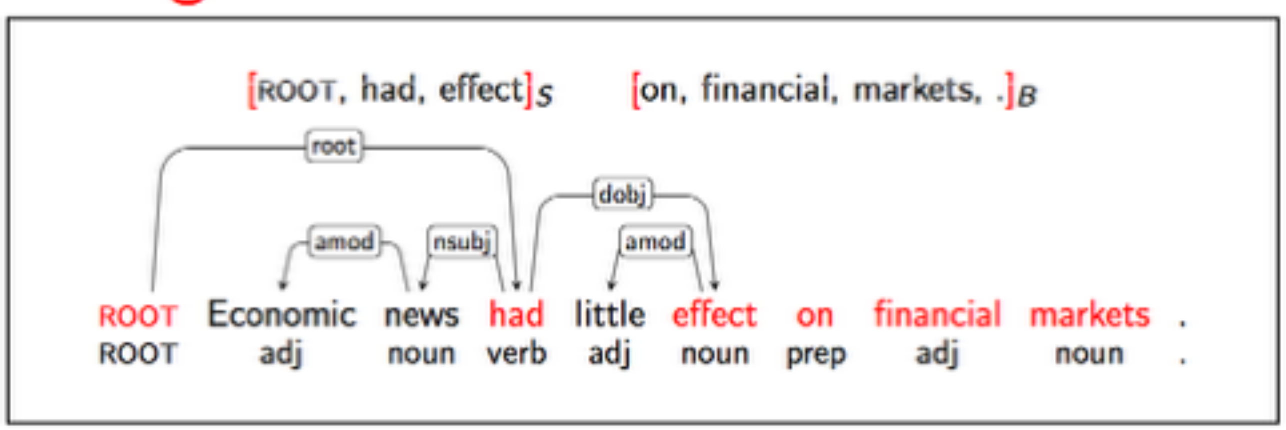
Features

$$
\begin{aligned}
\text{pos}(S_2) &= \text{ROOT} \\
\text{pos}(S_1) &= \text{verb} \\
\text{pos}(S_0) &= \text{noun} \\
\text{pos}(B_0) &= \text{prep} \\
\text{pos}(B_1) &= \text{adj} \\
\text{pos}(B_2) &= \text{noun}
\end{aligned}
$$

Static Lexical Features

- Features over input tokens relative to S and B

Configuration



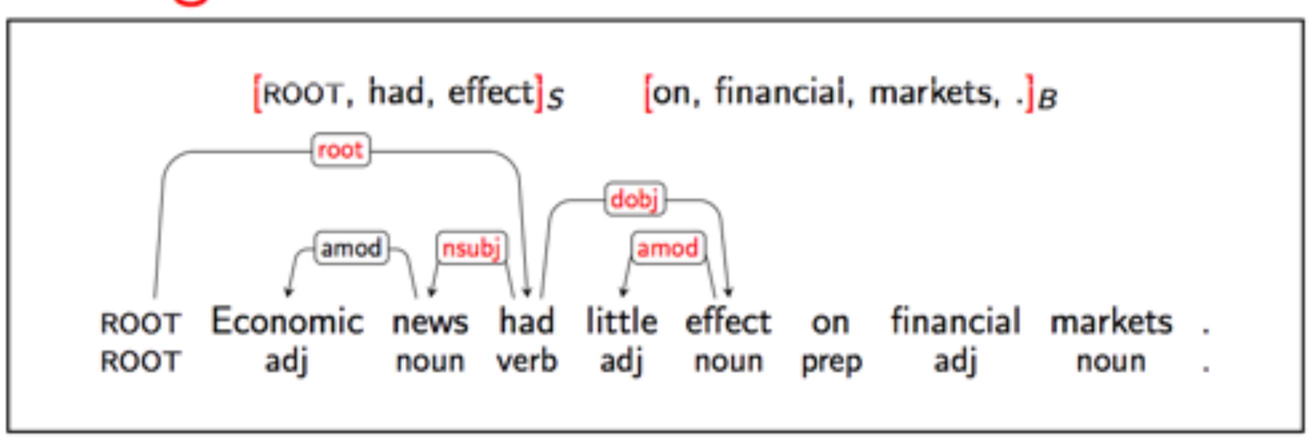
Features

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

Dynamic Syntactic Features

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

Configuration



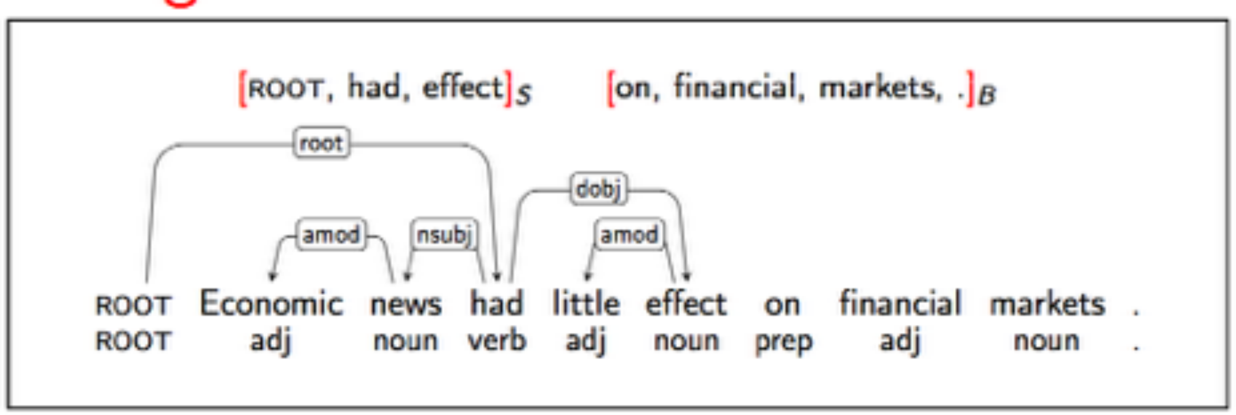
Features

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

Dynamic Historical 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

$$t_{i-1} = \text{Right-Arc(dobj)}$$
$$t_{i-2} = \text{Left-Arc(amod)}$$
$$t_{i-3} = \text{Shift}$$
$$t_{i-4} = \text{Right-Arc(root)}$$
$$t_{i-5} = \text{Left-Arc(nsubj)}$$
$$t_{i-6} = \text{Shift}$$

- Feature representation unconstrained by parsing algorithm

Greedy Local Transition-Based Parsing

- Advantages:
  - Highly efficient parsing – linear time complexity with constant time oracles and transitions
  - Rich history-based feature representations – no rigid constraints from inference algorithm
- Drawback:
  - Sensitive to search errors and error propagation due to greedy inference and local learning
- The major question in transition-based parsing has been how to improve learning and inference, while maintaining high efficiency and rich feature models

Evaluation Metrics

- Standard setup:
  - Test set $\mathcal{E} = \{(x_1, G_1), (x_2, G_2), \ldots, (x_n, G_n)\}$
  - Parser predictions $\mathcal{P} = \{(x_1, G'_1), (x_2, G'_2), \ldots, (x_n, G'_n)\}$
- Evaluation on the word (arc) level:
  - Labeled attachment score (LAS) $=$ head and label
  - Unlabeled attachment score (UAS) $=$ head
  - Label accuracy (LA) $=$ label
- Evaluation on the sentence (graph) level:
  - Exact match (labeled or unlabeled) $=$ complete graph
- NB: Evaluation metrics may or may not include punctuation

Multilingual Dependency Parsing

Parsing accuracy for 7 parsers on 13 languages, measured by labeled attachment score (LAS), unlabeled attachment score (UAS) and label accuracy (LA). NP-L = non-projective list-based; P-L = projective list-based; PP-L = pseudo-projective list-based; P-E = projective arc-eager stack-based; PP-E = pseudo-projective arc-eager stack-based; P-S = projective arc-standard stack-based; PP-S = pseudo-projective arc-standard stack-based; McD = McDonald, Lerman and Pereira (2006); Niv = Nivre et al. (2006).

	Labeled Attachment Score (LAS)								
Language	**NP-L**	**P-L**	**PP-L**	**P-E**	**PP-E**	**P-S**	**PP-S**	**McD**	**Niv**
Arabic	63.25	63.19	63.13	64.93	64.95	65.79	**66.05**	66.91	66.71
Bulgarian	**87.79**	87.75	87.39	87.75	87.41	86.42	86.71	87.57	87.41
Chinese	85.77	85.96	85.96	85.96	85.96	**86.00**	**86.00**	85.90	86.92
Czech	78.12	76.24	78.04	76.34	77.46	78.18	**80.12**	80.18	78.42
Danish	**84.59**	84.15	84.35	84.25	84.45	84.17	84.15	84.79	84.77
Dutch	**77.41**	74.71	76.95	74.79	76.89	73.27	74.79	79.19	78.59
German	84.42	84.21	84.38	84.23	84.46	**84.58**	**84.58**	87.34	85.82
Japanese	**90.97**	90.57	90.53	90.83	90.89	90.59	90.63	90.71	91.65
Portuguese	**86.70**	85.91	86.20	85.83	86.12	85.39	86.09	86.82	87.60
Slovene	70.06	69.88	70.12	69.50	70.22	**72.00**	71.88	73.44	70.30
Spanish	**80.18**	79.80	79.60	79.84	79.60	78.70	78.42	82.25	81.29
Swedish	**83.03**	82.63	82.41	82.63	82.41	82.12	81.54	82.55	84.58
Turkish	64.69	64.49	64.37	64.37	64.43	64.67	**64.73**	63.19	65.68
Average	**79.77**	79.19	79.49	79.33	79.63	79.38	79.67	80.83	80.75