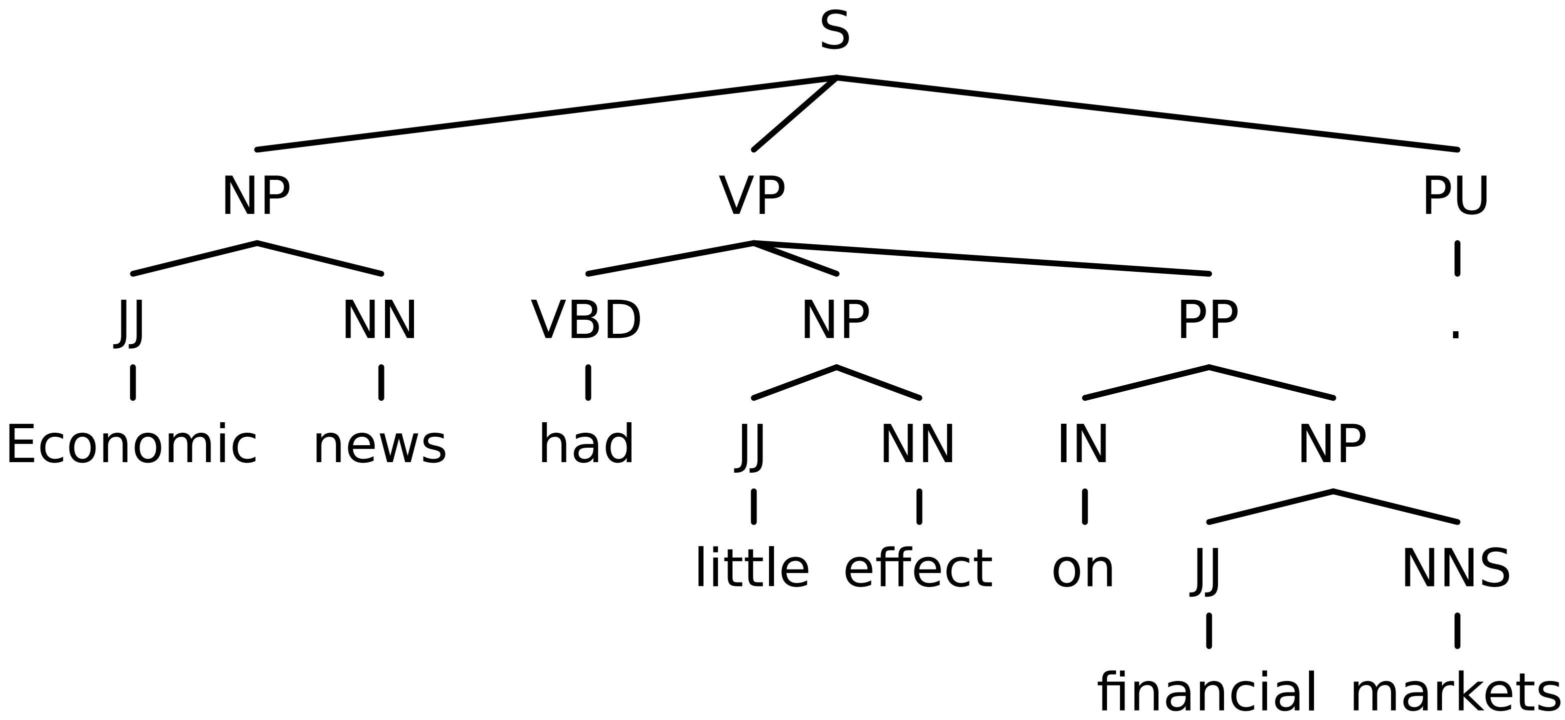
Acknowledgements

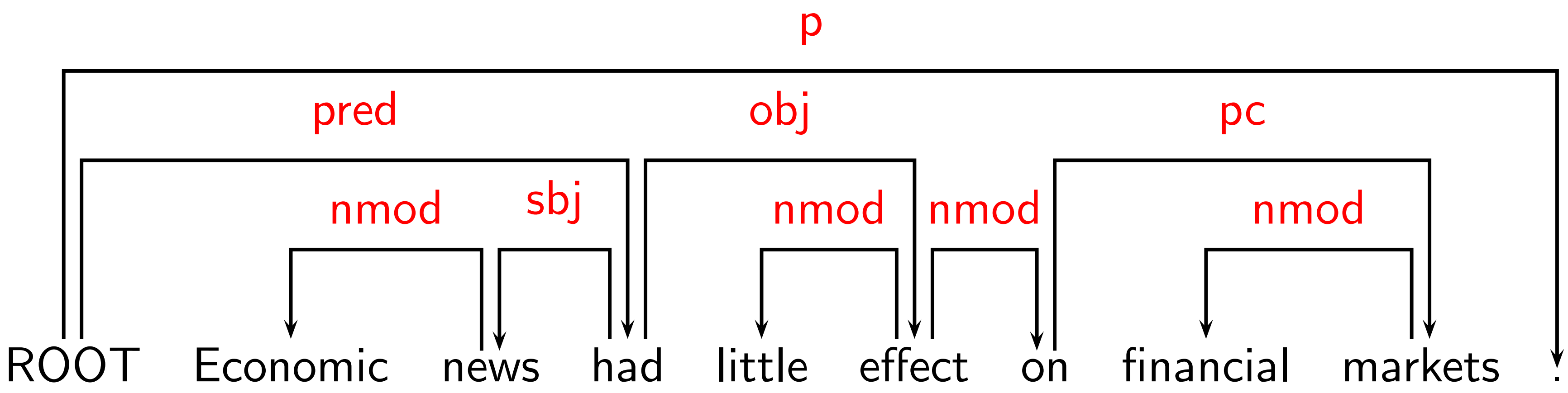
- These slides are heavily inspired by
  - an ESSLLI 2007 course by Joakim Nivre and Ryan McDonald
  - an ACL-COLING tutorial by Joakim Nivre and Sandra Kübler

Phrase-Structure Trees



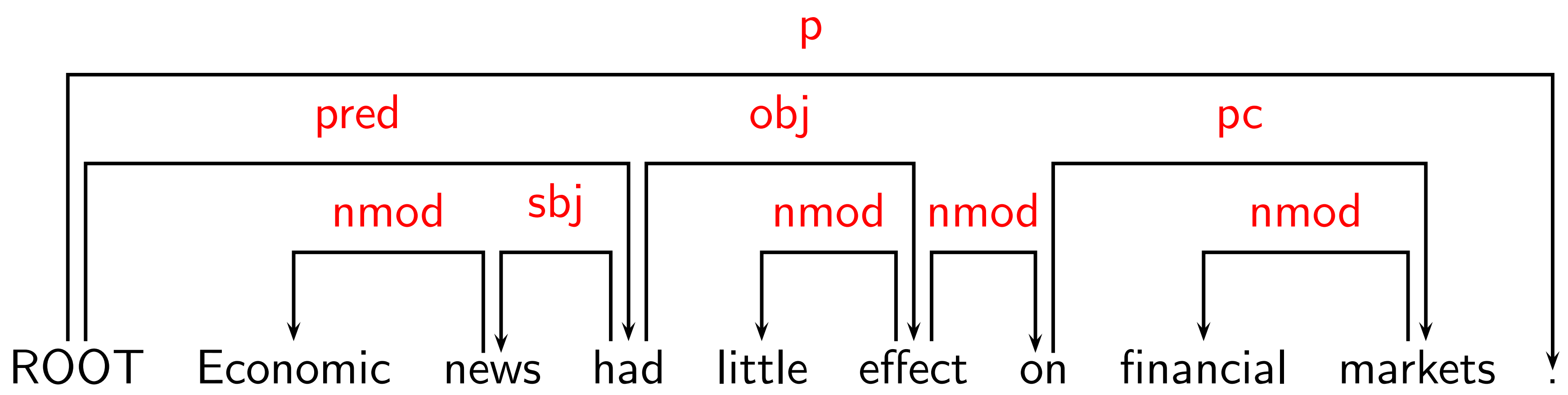
Dependency Trees

- **Basic idea:**
  - Syntactic structure = lexical items linked by relations
  - Syntactic structures are usually trees (… but not always)
- Relations H → D
  - H is the head (or governor)
  - D is the dependent



Dependency Trees

- **Parsers**
  - are easy to implement and evaluate
- **Dependency-based representations**
  - are suitable for free word order languages
  - are often close to the predicate argument structure



Dependency Trees

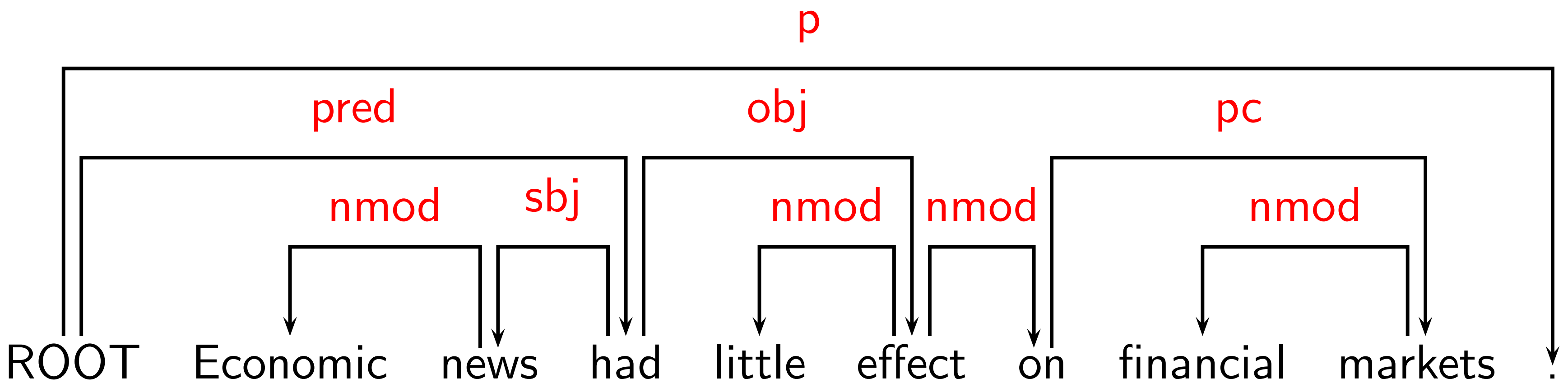
- Some criteria for dependency relations between a head H and a dependent D in a linguistic construction C:
  - H determines the syntactic category of C; H can replace C.
  - H determines the semantic category of C; D specifies H.
  - H is obligatory; D may be optional.
  - H selects D and determines whether D is obligatory.
  - The form of D depends on H (agreement or government).
  - The linear position of D is specified with reference to H.

Dependency Trees

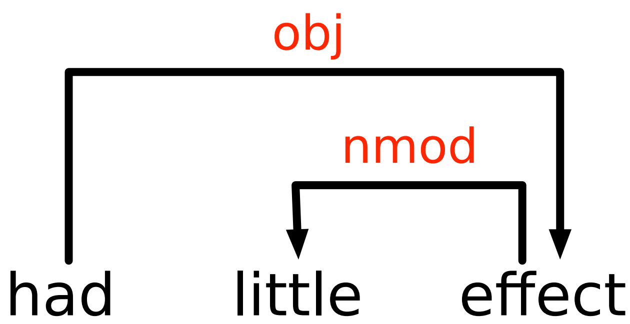
- Clear cases:
  - Subject, Object, ...
- Less clear cases:
  - complex verb groups
  - subordinate clauses
  - coordination
  - ...

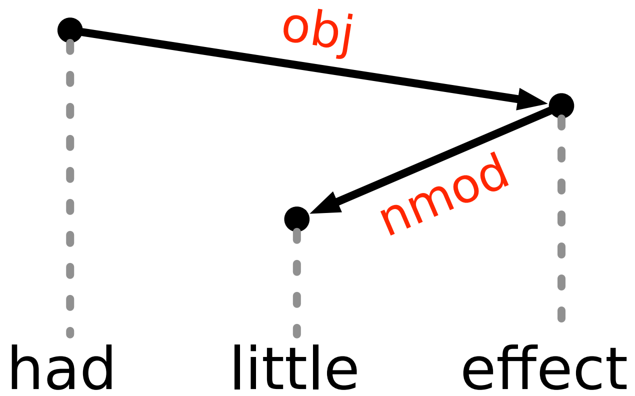
Dependency Graphs

- Graph G = ⟨V, A, L, <⟩
  - V = a set of vertices (nodes)
  - A = a set of arcs (directed edges)
  - L = a set of edge labels
  - < = a linear order on V



Dependency Trees – Notation



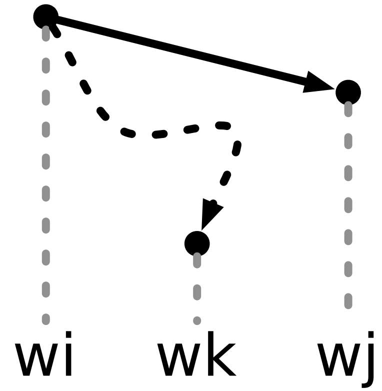


Dependency Graphs / Trees

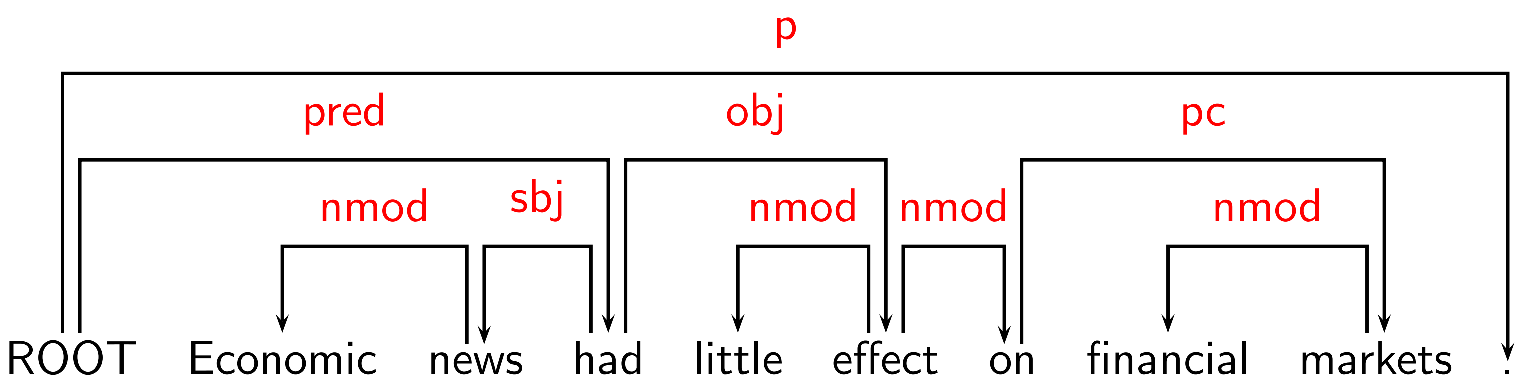
- Formal conditions on dependency graphs:
  - G is weakly connected
  - G is acyclic
  - Every node in G has at most one head
  - G is projective

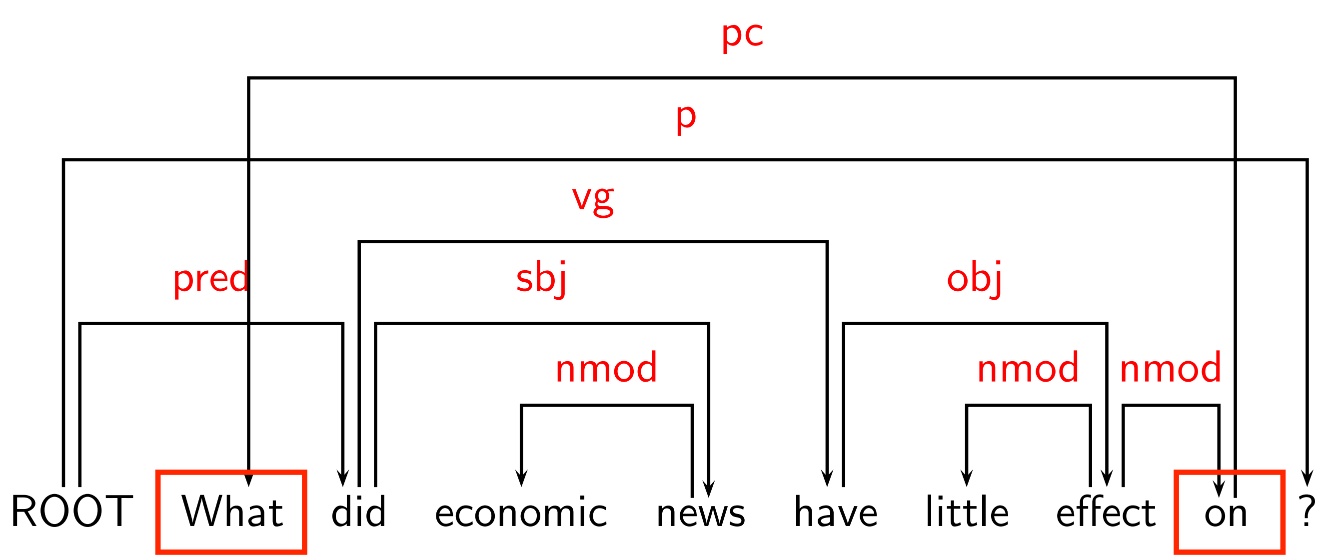
Projectivity

- A dependency graph G is projective iff
  - if $w_i \rightarrow w_j$, then $w_i \rightarrow^* w_k$ for all $w_i < w_k < w_j$ or $w_j < w_k < w_i$
  - if w_i is the head of w_j, then there must be a directed path from w_i to w_k, for all w_k between w_i and w_j.
- We need non-projectivity for
  - long distance dependencies
  - free word order



Projectivity





Projectivity

Language	Sentences	Dependencies
Arabic [Maamouri and Bies 2004]	11.2%	0.4%
Basque [Aduriz et al. 2003]	26.2%	2.9%
Czech [Hajic et al. 2001]	23.2%	1.9%
Danish [Kromann 2003]	15.6%	1.0%
Greek [Prokopidis et al. 2005]	20.3%	1.1%
Russian [Boguslavsky et al. 2000]	10.6%	0.9%
Slovenian [Dzeroski et al. 2006]	22.2%	1.9%
Turkish [Oflazer et al. 2003]	11.6%	1.5%

Dependency-based Parsing

- Grammar-based
- Data-driven
  - Transition-based
  - Graph-based

Transition-based Parsing

- Configurations $\langle S, Q, A\rangle$
  - S = a stack of partially processed tokens (nodes)
  - Q = a queue of unprocessed input tokens
  - A = a set of dependency arcs
- Initial configuration for input $w_1 \ldots w_n$
  - $\langle [w_0], [w_1, \ldots, w_n], \{\}\rangle$, w_0 = ROOT
- Terminal (accepting) configuration
  - $\langle \ldots, [], \ldots \rangle$

Transitions („Arc-Standard“)

- Left-Arc(r)
  - adds a dependency arc (w_j , r, w_i) to the arc set A, where w_i is the word on top of the stack and w_j is the first word in the buffer, and pops the stack.
- Right-Arc(r)
  - adds a dependency arc (w_i , r, w_j) to the arc set A, where w_i is the word on top of the stack and w_j is the first word in the buffer, pops the stack and replaces w_j by w_i at the head of buffer.

Transitions („Arc-Standard“)

- Left-Arc(r)

$$\frac{\langle[\ldots, w_i], [w_j, \ldots], A\rangle}{\langle[\ldots], [w_j, \ldots], A \cup \{(w_j, r, w_i)\}\rangle} \quad i \neq 0, \neg\exists k \exists l' \, (w_k, l', w_i) \in A$$

- Right-Arc(r)

$$\frac{\langle[\ldots, w_i], [w_j, \ldots], A\rangle}{\langle[\ldots], [w_i, \ldots], A \cup \{(w_i, r, w_j)\}\rangle} \quad \neg\exists k \exists l' \, (w_k, l', w_j) \in A$$

- Shift

$$\frac{\langle[\ldots], [w_i, \ldots], A\rangle}{\langle[\ldots, w_i], [\ldots], A\rangle}$$

An Example

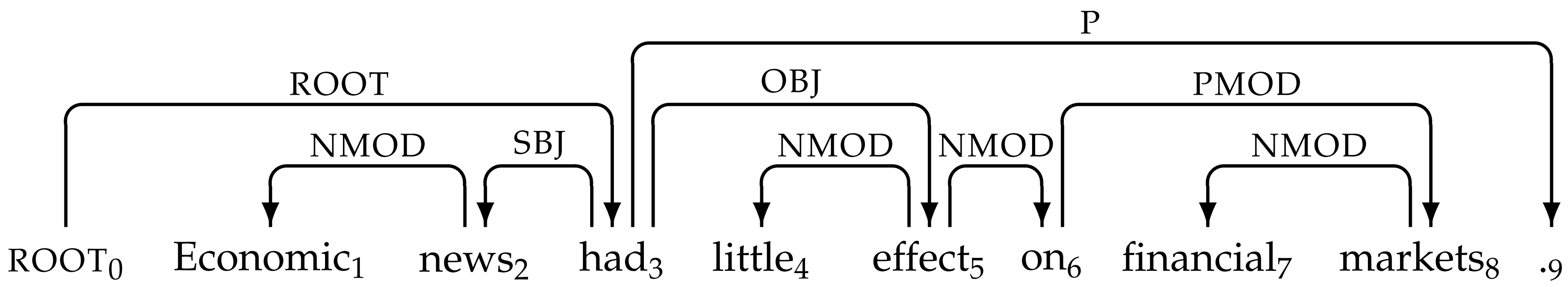


Figure 2
Dependency graph for an English sentence from the Penn Treebank.

An Example

	(	$[0]$,	$[1, \ldots, 9]$,	$\emptyset$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 1]$,	$[2, \ldots, 9]$,	$\emptyset$)
$\textsc{Left-Arc}_{\textsc{nmod}} \Longrightarrow$	(	$[0]$,	$[2, \ldots, 9]$,	$A_1 = \{(2, \textsc{nmod}, 1)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 2]$,	$[3, \ldots, 9]$,	A_1)
$\textsc{Left-Arc}_{\textsc{sbj}} \Longrightarrow$	(	$[0]$,	$[3, \ldots, 9]$,	$A_2 = A_1 \cup \{(3, \textsc{sbj}, 2)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 3]$,	$[4, \ldots, 9]$,	A_2)
$\textsc{Shift} \Longrightarrow$	(	$[0, 3, 4]$,	$[5, \ldots, 9]$,	A_2)
$\textsc{Left-Arc}_{\textsc{nmod}} \Longrightarrow$	(	$[0, 3]$,	$[5, \ldots, 9]$,	$A_3 = A_2 \cup \{(5, \textsc{nmod}, 4)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 3, 5]$,	$[6, \ldots, 9]$,	A_3)
$\textsc{Shift} \Longrightarrow$	(	$[0, \ldots 6]$,	$[7, 8, 9]$,	A_3)
$\textsc{Shift} \Longrightarrow$	(	$[0, \ldots, 7]$,	$[8, 9]$,	A_3)
$\textsc{Left-Arc}_{\textsc{nmod}} \Longrightarrow$	(	$[0, \ldots 6]$,	$[8, 9]$,	$A_4 = A_3 \cup \{(8, \textsc{nmod}, 7)\}$)
$\textsc{Right-Arc}^{s}_{\textsc{pmod}} \Longrightarrow$	(	$[0, 3, 5]$,	$[6, 9]$,	$A_5 = A_4 \cup \{(6, \textsc{pmod}, 8)\}$)
$\textsc{Right-Arc}^{s}_{\textsc{nmod}} \Longrightarrow$	(	$[0, 3]$,	$[5, 9]$,	$A_6 = A_5 \cup \{(5, \textsc{nmod}, 6)\}$)
$\textsc{Right-Arc}^{s}_{\textsc{obj}} \Longrightarrow$	(	$[0]$,	$[3, 9]$,	$A_7 = A_6 \cup \{(3, \textsc{obj}, 5)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 3]$,	$[9]$,	A_7)
$\textsc{Right-Arc}^{s}_{\textsc{p}} \Longrightarrow$	(	$[0]$,	$[3]$,	$A_8 = A_7 \cup \{(3, \textsc{p}, 9)\}$)
$\textsc{Right-Arc}^{s}_{\textsc{root}} \Longrightarrow$	(	$[\,]$,	$[0]$,	$A_9 = A_8 \cup \{(0, \textsc{root}, 3)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0]$,	$[\,]$,	A_9)

Deterministic Parsing

- oracle(c):
  - predicts the next transition
- parse(w_1 … w_n):
  - c := $\langle[w_0 = \text{ROOT}], [w_1, \ldots, w_n], \{\}\rangle$
  - while c is not terminal
    - t := oracle(c)
    - c := t(c)
  - return G = $\langle\{w_0, \ldots, w_n\}, A_c\rangle$

Deterministic Parsing

- Linear time complexity: the algorithm terminates after 2n steps for input sentences with n words.
- The algorithm is complete and correct for the class of projective dependency trees:
  - For every projective dependency tree T there is a sequence of transitions that generates T
  - Every sequence of transition steps generates a projective dependency tree
- Whether the resulting dependency tree is correct or not depends of course on the oracle.

The oracle

- Approximate the oracle by a classifier
- Represent configurations be feature vectors; for instance
  - lexical properties (word form, lemma)
  - category (part of speech)
  - labels of partial dependency trees
  - ...

An Example

$f(c_0)$	=	(ROOT	Economic	news	NULL	NULL	NULL	NULL)
$f(c_1)$	=	(Economic	news	had	NULL	NULL	NULL	NULL)
$f(c_2)$	=	(ROOT	news	had	NULL	NULL	ATT	NULL)
$f(c_3)$	=	(news	had	little	ATT	NULL	NULL	NULL)
$f(c_4)$	=	(ROOT	had	little	NULL	NULL	SBJ	NULL)
$f(c_5)$	=	(had	little	effect	SBJ	NULL	NULL	NULL)
$f(c_6)$	=	(little	effect	on	NULL	NULL	NULL	NULL)
$f(c_7)$	=	(had	effect	on	SBJ	NULL	ATT	NULL)
$f(c_8)$	=	(effect	on	financial	ATT	NULL	NULL	NULL)
$f(c_9)$	=	(on	financial	markets	NULL	NULL	NULL	NULL)
$f(c_{10})$	=	(financial	markets	.	NULL	NULL	NULL	NULL)
$f(c_{11})$	=	(on	markets	.	NULL	NULL	ATT	NULL)
$f(c_{12})$	=	(effect	on	.	ATT	NULL	NULL	ATT)
$f(c_{13})$	=	(had	effect	.	SBJ	NULL	ATT	ATT)
$f(c_{14})$	=	(ROOT	had	.	NULL	NULL	SBJ	OBJ)
$f(c_{15})$	=	(had	.	NULL	SBJ	OBJ	NULL	NULL)
$f(c_{16})$	=	(ROOT	had	NULL	NULL	NULL	SBJ	PU)
$f(c_{17})$	=	(NULL	ROOT	NULL	NULL	NULL	NULL	PRED)
$f(c_{18})$	=	(ROOT	NULL	NULL	NULL	PRED	NULL	NULL)

Non-projective Parsing

- Configurations $\langle L_1, L_2, Q, A \rangle$
  - L_1, L_2 are stacks of partially processed nodes
  - Q = a queue of unprocessed input tokens
  - A = a set of dependency arcs
- Initial configuration for input $w_1 \ldots w_n$
  - $\langle [w_0], [], [w_1, \ldots, w_n], \{\} \rangle$, w_0 = ROOT
- Terminal configuration:
  - $\langle [w_0, w_1, \ldots, w_n], [], [], A \rangle$

Transitions

- Left-Arc(l)

$$\frac{\langle [\ldots, w_i], [\ldots], [w_j, \ldots], A \rangle}{\langle [\ldots], [w_i, \ldots], [w_j, \ldots], A \cup \{(w_j, l, w_i)\} \rangle}$$

$i \neq 0$

$\neg \exists k \exists l' \; (w_k, l', w_i) \in A$

$\neg \; w_i \rightarrow^* w_j$

- Right-Arc(l)

$$\frac{\langle [\ldots, w_i], [\ldots], [w_j, \ldots], A \rangle}{\langle [\ldots], [w_i, \ldots], [w_j, \ldots], A \cup \{(w_i, l, w_j)\} \rangle}$$

$\neg \exists k \exists l' \; (w_k, l', w_j) \in A$

$\neg \; w_i \rightarrow^* w_j$

Transitions

- No-Arc

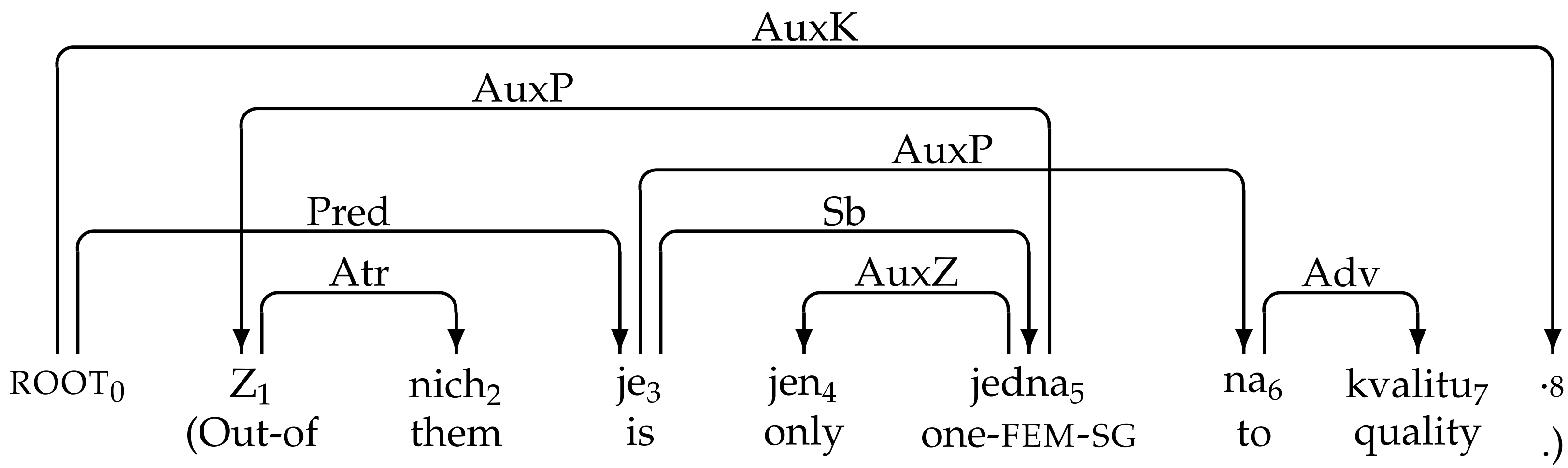
$$\frac{\langle [\ldots, w_i], [\ldots], [\ldots], A\rangle}{\langle [\ldots], [w_i, \ldots], [\ldots], A\rangle}$$

- Shift

$$\frac{\langle [\ldots]_{L1}, [\ldots]_{L2}, [w_i, \ldots], A\rangle}{\langle [\ldots]_{L1} \bullet [\ldots, w_i]_{L2}, [], [\ldots], A\rangle}$$

- $L_1 \bullet L_2$ = the concatenation of L_1 and L_2

An Example



("Only one of them concerns quality.")

Figure 1
Dependency graph for a Czech sentence from the Prague Dependency Treebank.

An Example

$$
\begin{array}{rll}
 & (& [0], \quad [\,], \quad [1,\ldots,8], \quad \emptyset \quad) \\
\textsc{Shift}^{\lambda} & \Longrightarrow (& [0,1], \quad [\,], \quad [2,\ldots,8], \quad \emptyset \quad) \\
\textsc{Right-Arc}^{n}_{\text{Atr}} & \Longrightarrow (& [0], \quad [1], \quad [2,\ldots,8], \quad A_1 = \{(1,\text{Atr},2)\} \quad) \\
\textsc{Shift}^{\lambda} & \Longrightarrow (& [0,1,2], \quad [\,], \quad [3,\ldots,8], \quad A_1 \quad) \\
\textsc{No-Arc}^{n} & \Longrightarrow (& [0,1], \quad [2], \quad [3,\ldots,8], \quad A_1 \quad) \\
\textsc{No-Arc}^{n} & \Longrightarrow (& [0], \quad [1,2], \quad [3,\ldots,8], \quad A_1 \quad) \\
\textsc{Right-Arc}^{n}_{\text{Pred}} & \Longrightarrow (& [\,], \quad [0,1,2], \quad [3,\ldots,8], \quad A_2 = A_1 \cup \{(0,\text{Pred},3)\} \quad) \\
\textsc{Shift}^{\lambda} & \Longrightarrow (& [0,\ldots,3], \quad [\,], \quad [4,\ldots,8], \quad A_2 \quad) \\
\textsc{Shift}^{\lambda} & \Longrightarrow (& [0,\ldots,4], \quad [\,], \quad [5,\ldots,8], \quad A_2 \quad) \\
\textsc{Left-Arc}^{n}_{\text{AuxZ}} & \Longrightarrow (& [0,\ldots,3], \quad [4], \quad [5,\ldots,8], \quad A_3 = A_2 \cup \{(5,\text{AuxZ},4)\} \quad) \\
\textsc{Right-Arc}^{n}_{\text{Sb}} & \Longrightarrow (& [0,1,2], \quad [3,4], \quad [5,\ldots,8], \quad A_4 = A_3 \cup \{(3,\text{Sb},5)\} \quad) \\
\textsc{No-Arc}^{n} & \Longrightarrow (& [0,1], \quad [2,3,4], \quad [5,\ldots,8], \quad A_4 \quad) \\
\textsc{Left-Arc}^{n}_{\text{AuxP}} & \Longrightarrow (& [0], \quad [1,\ldots,4], \quad [5,\ldots,8], \quad A_5 = A_4 \cup \{(5,\text{AuxP},1)\} \quad) \\
\textsc{Shift}^{\lambda} & \Longrightarrow (& [0,\ldots,5], \quad [\,], \quad [6,7,8], \quad A_5 \quad) \\
\textsc{No-Arc}^{n} & \Longrightarrow (& [0,\ldots,4], \quad [5], \quad [6,7,8], \quad A_5 \quad)
\end{array}
$$

An Example

$$
\begin{array}{rcllllll}
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,4], & [5], & [6,7,8], & A_5 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,3], & [4,5], & [6,7,8], & A_5 &) \\
\textsc{Right-Arc}^{n}_{\text{AuxP}} & \Longrightarrow & (& [0,1,2], & [3,4,5], & [6,7,8], & A_6 = A_5 \cup \{(3, \text{AuxP}, 6)\} &) \\
\textsc{Shift}^{\lambda} & \Longrightarrow & (& [0,\ldots,6], & [\,], & [7,8], & A_6 &) \\
\textsc{Right-Arc}^{n}_{\text{Adv}} & \Longrightarrow & (& [0,\ldots,5], & [6], & [7,8], & A_7 = A_6 \cup \{(6, \text{Adv}, 7)\} &) \\
\textsc{Shift}^{\lambda} & \Longrightarrow & (& [0,\ldots,7], & [\,], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,6], & [7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,5], & [6,7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,4], & [5,6,7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,\ldots,3], & [4,\ldots,7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,1,2], & [3,\ldots,7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0,1], & [2,\ldots,7], & [8], & A_7 &) \\
\textsc{No-Arc}^{n} & \Longrightarrow & (& [0], & [1,\ldots,7], & [8], & A_7 &) \\
\textsc{Right-Arc}^{n}_{\text{AuxK}} & \Longrightarrow & (& [\,], & [0,\ldots,7], & [8], & A_8 = A_7 \cup \{(0, \text{AuxK}, 8)\} &) \\
\textsc{Shift}^{\lambda} & \Longrightarrow & (& [0,\ldots,8], & [\,], & [\,], & A_8 &)
\end{array}
$$

Non-projective Parsing

- The algorithm is sound and complete for the class of dependency forests
- Time complexity is $O(n^2)$
  - at most n Shift-transitions
  - between the i-th and (i+1)-th Shift-transition there are at most i transitions (left-arc, right-arc, no-arc)

Literature

- Sandra Kübler, Ryan McDonald and Joakim Nivre (2009). Dependency Parsing.
- Joakim Nivre (2008). Algorithms for Deterministic Incremental Dependency Parsing. Computational Linguistics 34(4), 513–553.