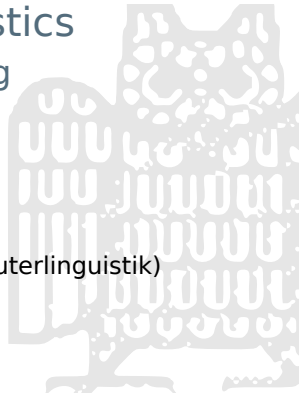


Computational Linguistics

Dependency-based Parsing

Clayton Greenberg
Stefan Thater

FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes
Summer 2015

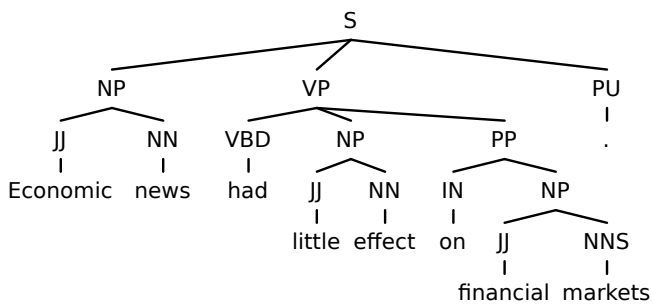


Acknowledgements

- 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

2

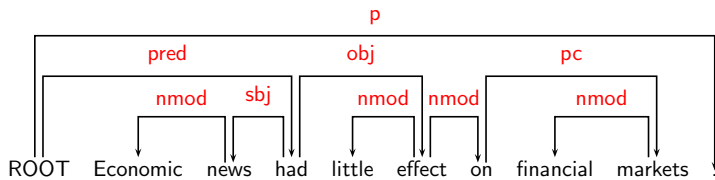
Phrase-Structure Trees



3

Dependency Trees

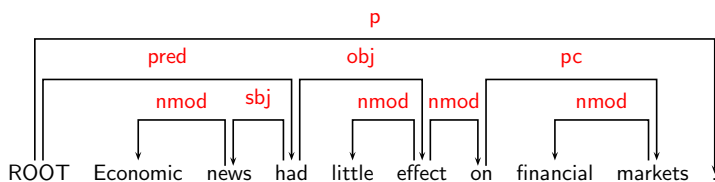
- **Basic idea:**
 - Syntactic structure = lexical items linked by relations
 - Syntactic structures are usually trees (... but not always)
- Relations $H \rightarrow D$
 - H is the head (or governor)
 - D is the dependent



4

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



5

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.

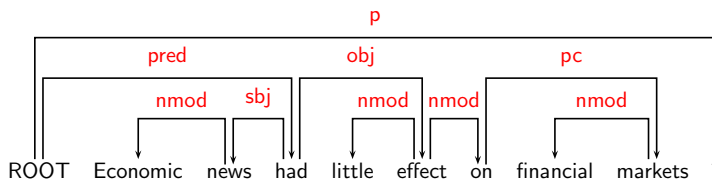
6

Dependency Trees

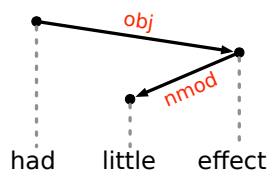
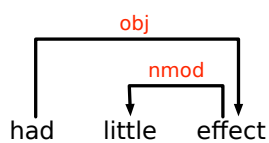
- Clear cases:
 - Subject, Object, ...
- Less clear cases:
 - complex verb groups
 - subordinate clauses
 - coordination
 - ...

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

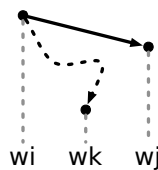


Dependency Graphs / Trees

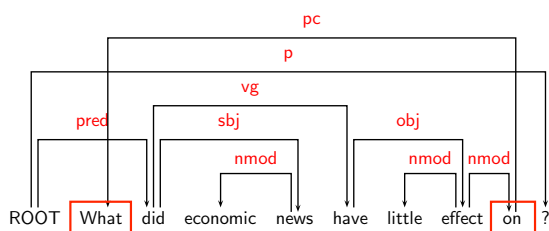
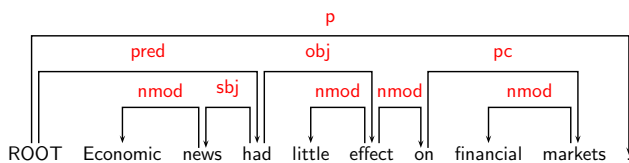
- Formal conditions on dependency graphs:
 - G is weakly connected
 - G is acyclic
 - Every node in G has at most one head
 - G is projective

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%

13

Dependency-based Parsing

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

14

Transition-based Parsing

- Configurations (S, Q, A)
 - 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 $w_1 \dots w_n$
 - $\langle [w_0], [w_1, \dots, w_n], \{\} \rangle$, $w_0 = \text{ROOT}$
- Terminal (accepting) configuration
 - $\langle \dots, [], \dots \rangle$

15

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.

16

Transitions („Arc-Standard“)

- Left-Arc(r)

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

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

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

17

An Example

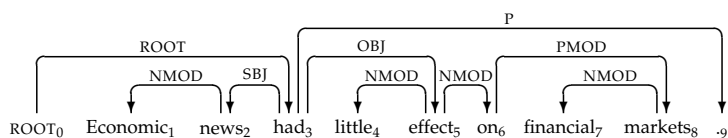


Figure 2
Dependency graph for an English sentence from the Penn Treebank.

18

An Example

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

19

Deterministic Parsing

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

20

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.

21

The oracle

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

22

An Example

```
f(c0) = (root Economic news NULL NULL NULL NULL)
f(c1) = (Economic news had NULL NULL NULL NULL)
f(c2) = (root news had NULL NULL ATT NULL)
f(c3) = (news had little ATT NULL NULL NULL)
f(c4) = (root had little NULL NULL SBJ NULL)
f(c5) = (had little effect SBJ NULL NULL NULL)
f(c6) = (little effect on NULL NULL NULL NULL)
f(c7) = (had effect on SBJ NULL ATT NULL)
f(c8) = (effect on financial ATT NULL NULL NULL)
f(c9) = (on financial markets NULL NULL NULL NULL)
f(c10) = (financial markets . NULL NULL NULL NULL)
f(c11) = (on markets . NULL NULL ATT NULL)
f(c12) = (effect on . ATT NULL NULL ATT)
f(c13) = (had effect . SBJ NULL ATT ATT)
f(c14) = (root had . NULL NULL SBJ OBJ)
f(c15) = (had . NULL SBJ OBJ NULL NULL)
f(c16) = (root had NULL NULL NULL SBJ PU)
f(c17) = (NULL ROOT NULL NULL NULL NULL PRED)
f(c18) = (root NULL NULL NULL PRED NULL NULL)
```

23

Non-projective Parsing

- Configurations (L_1, L_2, Q, A)
 - 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 $w_1 \dots w_n$
 - $\langle [w_0], [], [w_1, \dots, w_n], \{\} \rangle$, $w_0 = \text{ROOT}$
- Terminal configuration:
 - $\langle [w_0, w_1, \dots, w_n], [], [], A \rangle$

24

Transitions

- Left-Arc(l)

$$\frac{\langle [\dots, w_i], [\dots], [w_j, \dots], A \rangle}{\langle [\dots], [w_i, \dots], [w_j, \dots], A \cup \{(w_j, l, w_i)\} \rangle} \quad \begin{array}{l} i \neq 0 \\ \neg \exists k \exists l' (w_k, l', w_i) \in A \\ \neg w_i \rightarrow^* w_j \end{array}$$

- Right-Arc(l)

$$\frac{\langle [\dots, w_i], [\dots], [w_j, \dots], A \rangle}{\langle [\dots], [w_i, \dots], [w_j, \dots], A \cup \{(w_i, l, w_j)\} \rangle} \quad \begin{array}{l} \neg \exists k \exists l' (w_k, l', w_j) \in A \\ \neg w_i \rightarrow^* w_j \end{array}$$

25

Transitions

- No-Arc

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

- Shift

$$\frac{\langle [\dots]_{L_1}, [\dots]_{L_2}, [w_i, \dots], A \rangle}{\langle [\dots]_{L_1} \cdot [\dots, w_i]_{L_2}, [], [\dots], A \rangle}$$

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

26

An Example

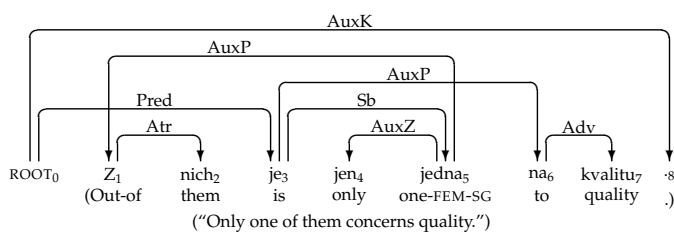


Figure 1
Dependency graph for a Czech sentence from the Prague Dependency Treebank.

27

An Example

	([0, [1, [1, [1, ..., 8], 0)
SHIFT ^λ	⇒ ([0, 1], [1, [2, ..., 8], 0)
RIGHT-ARC _{Atr} ⁿ	⇒ ([0], [1], [2, ..., 8], A ₁ = {(1, Atr, 2)})
SHIFT ^λ	⇒ ([0, 1, 2], [1, [3, ..., 8], A ₁)
NO-ARC ⁿ	⇒ ([0, 1], [2], [3, ..., 8], A ₁)
NO-ARC ⁿ	⇒ ([0], [1, 2], [3, ..., 8], A ₁)
RIGHT-ARC _{Pred} ⁿ	⇒ ([], [0, 1, 2], [3, ..., 8], A ₂ = A ₁ ∪ {(0, Pred, 3)})
SHIFT ^λ	⇒ ([0, ..., 3], [1, [4, ..., 8], A ₂)
SHIFT ^λ	⇒ ([0, ..., 4], [1, [5, ..., 8], A ₂)
LEFT-ARC _{AuxZ} ⁿ	⇒ ([0, ..., 3], [4], [5, ..., 8], A ₃ = A ₂ ∪ {(5, AuxZ, 4)})
RIGHT-ARC _{Sb} ⁿ	⇒ ([0, 1, 2], [3, 4], [5, ..., 8], A ₄ = A ₃ ∪ {(3, Sb, 5)})
NO-ARC ⁿ	⇒ ([0, 1], [2, 3, 4], [5, ..., 8], A ₄)
LEFT-ARC _{AuxP} ⁿ	⇒ ([0], [1, ..., 4], [5, ..., 8], A ₅ = A ₄ ∪ {(5, AuxP, 1)})
SHIFT ^λ	⇒ ([0, ..., 5], [1, [6, 7, 8], A ₅)
NO-ARC ⁿ	⇒ ([0, ..., 4], [5], [6, 7, 8], A ₅)

28

An Example

NO-ARC ⁿ	⇒ ([0, ..., 4], [5], [6, 7, 8], A ₅)
NO-ARC ⁿ	⇒ ([0, ..., 3], [4, 5], [6, 7, 8], A ₅)
RIGHT-ARC _{AuxP} ⁿ	⇒ ([0, 1, 2], [3, 4, 5], [6, 7, 8], A ₆ = A ₅ ∪ {(3, AuxP, 6)})
SHIFT ^λ	⇒ ([0, ..., 6], [1, [7, 8], A ₆)
RIGHT-ARC _{Adv} ⁿ	⇒ ([0, ..., 5], [6], [7, 8], A ₇ = A ₆ ∪ {(6, Adv, 7)})
SHIFT ^λ	⇒ ([0, ..., 7], [1, [8], A ₇)
NO-ARC ⁿ	⇒ ([0, ..., 6], [7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0, ..., 5], [6, 7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0, ..., 4], [5, 6, 7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0, ..., 3], [4, ..., 7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0, 1, 2], [3, ..., 7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0, 1], [2, ..., 7], [8], A ₇)
NO-ARC ⁿ	⇒ ([0], [1, ..., 7], [8], A ₇)
RIGHT-ARC _{AuxK} ⁿ	⇒ ([], [0, ..., 7], [8], A ₈ = A ₇ ∪ {(0, AuxK, 8)})
SHIFT ^λ	⇒ ([0, ..., 8], [1, [])

29

Non-projective Parsing

- The algorithm is sound and complete for the class of dependency forests
- Time complexity is O(n²)
 - 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)

30

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.