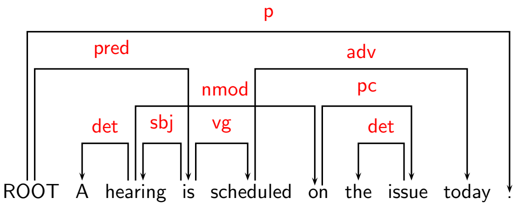
Sorting Out Dependency Parsing

Joakim Nivre

Uppsala University and Växjö University

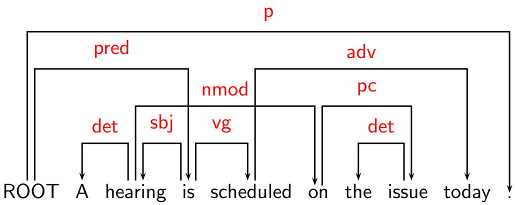
Introduction

- Syntactic parsing of natural language:
  - Who does what to whom?
- Dependency-based syntactic representations
  - have a natural way of representing discontinuous constructions,
  - give a transparent encoding of predicate-argument structure,
  - can be parsed using (simple) data-driven models,
  - can be parsed efficiently.



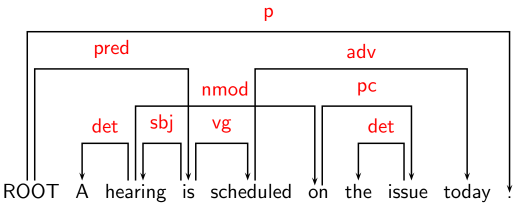
Introduction

- Syntactic parsing of natural language:
  - Who does what to whom?
- Dependency-based syntactic representations
  - have a natural way of representing discontinuous constructions,
  - give a transparent encoding of predicate-argument structure,
  - can be parsed using (simple) data-driven models,
  - can be parsed efficiently.



Introduction

- Syntactic parsing of natural language:
  - Who does what to whom?
- Dependency-based syntactic representations
  - have a natural way of representing discontinuous constructions,
  - give a transparent encoding of predicate-argument structure,
  - can be parsed using (simple) data-driven models,
  - can be parsed efficiently.



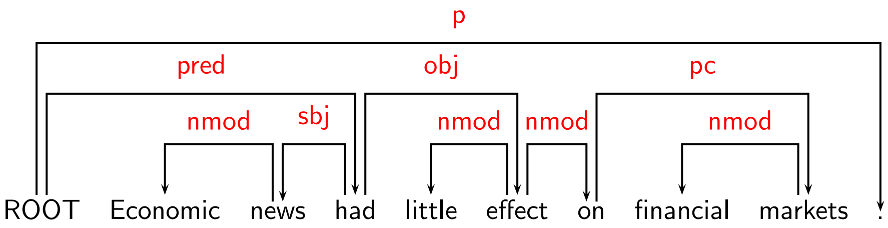
Structure of This Talk

- Part 1:
  - Transition-based dependency parsing
  - Restricted to projective structures
- Part 2:
  - Non-projective dependency parsing
  - Parsing = sorting + projective parsing

Dependency Parsing

Dependency Syntax

- The basic idea:
  - Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.
- Many different theoretical frameworks



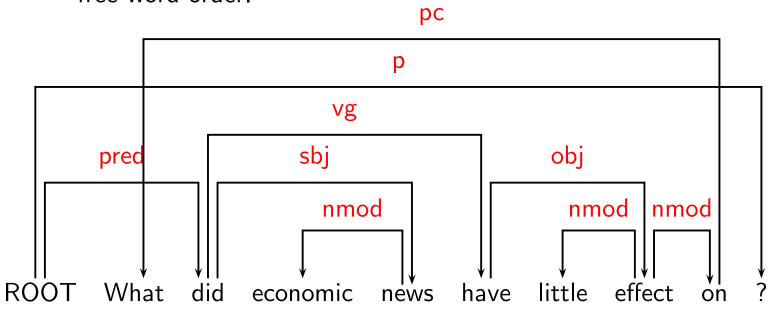
Dependency Trees

- A dependency structure is a labeled directed tree T, consisting of
  - a set V of nodes, labeled with words (including ROOT),
  - a set A of arcs, labeled with dependency types,
  - a linear precedence order $<$ on V,

  with the node labeled ROOT as the unique root.
- **Note:** Some frameworks do not assume that dependency structures are trees but allow general graphs.

Projectivity

- A dependency tree T is projective iff
  - for every arc $w_i \rightarrow w_j$ and every node w_k between w_i and w_j in the linear order, there is a (directed) path from w_i to w_k.
- Most theoretical frameworks do not assume projectivity.
- Non-projective structures are needed to account for
  - long-distance dependencies,
  - free word order.



Non-Projectivity in Natural Language

Language	Sentences	Dependencies
Arabic [Maamouri and Bies 2004]	11.2%	0.4%
Basque [Aduriz et al. 2003]	26.2%	2.9%
Czech [Hajič 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%
Slovene [Džeroski et al. 2006]	22.2%	1.9%
Turkish [Oflazer et al. 2003]	11.6%	1.5%

Data-Driven Dependency Parsing

- Dependency parsing based on (only) supervised learning from treebank data (annotated sentences)
- Graph-based [Eisner 1996, McDonald et al. 2005a]
  - Define a space of candidate dependency trees for a sentence
  - Learning: Induce a model for scoring an entire dependency tree for a sentence
  - Inference: Find the highest-scoring dependency tree, given the induced model
- Transition-based [Yamada and Matsumoto 2003, Nivre et al. 2004]:
  - Define a transition system (state machine) for mapping a sentence to its dependency tree
  - Learning: Induce a model for predicting the next state transition, given the transition history
  - Inference: Construct the optimal transition sequence, given the induced model

Transition-Based Dependency Parsing

Overview of the Approach

- The basic idea:
  - Define a transition system for dependency parsing
  - Train a classifier for predicting the next transition
  - Use the classifier to do parsing as greedy, deterministic search
- Advantages:
  - Efficient parsing (linear time complexity)
  - Robust disambiguation (discriminative classifiers)

Transition System: Configurations

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

  $([w_0]_S, [w_1, \ldots, w_n]_Q, \{\,\})$ **NB:** $w_0 = \mathrm{ROOT}$
- Termination:

  $([w_0]_S, [\,]_Q, A)$

Transition System: Transitions

- Left-Arc(l)

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

- Right-Arc(l)

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

- Shift

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

Deterministic Parsing

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

Parse($w_1, \ldots, w_n$)

$c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, \{\,\})$

while $Q_c \neq [\,]$ **or** $|S_c| > 1$

  $t \leftarrow o(c)$

  $c \leftarrow t(c)$

return $G = (\{w_0, w_1, \ldots, w_n\}, A_c)$

Example

$o(c) = \text{Shift}$

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

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Shift}$

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

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Left-Arc}_{nmod}$

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

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Shift}$

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

nmod

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Left-Arc}_{sbj}$

$[\![$ROOT news had$]\!]_S$ $[\![$little effect on financial markets .$]\!]_Q$

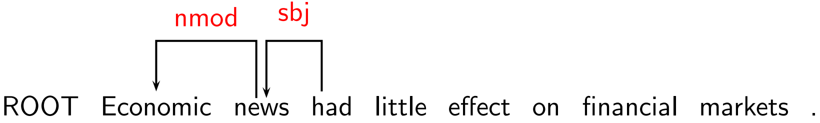
nmod

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Shift}$

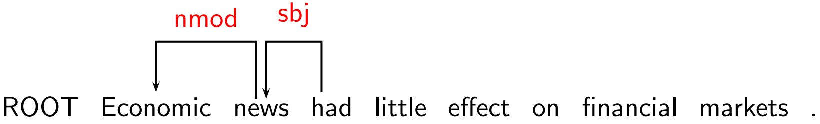
$[\![\text{ROOT} \quad \text{had}]\!]_S \quad [\![\text{little} \quad \text{effect} \quad \text{on} \quad \text{financial} \quad \text{markets} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

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



Example

$o(c) = \text{Left-Arc}_{nmod}$

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

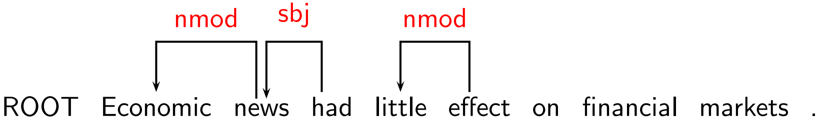
nmod sbj

ROOT Economic news had little effect on financial markets .

Example

$o(c) = \text{Shift}$

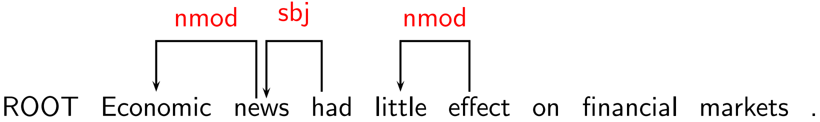
$[\![\text{ROOT} \quad \text{had} \quad \text{effect}]\!]_S \quad [\![\text{on} \quad \text{financial} \quad \text{markets} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

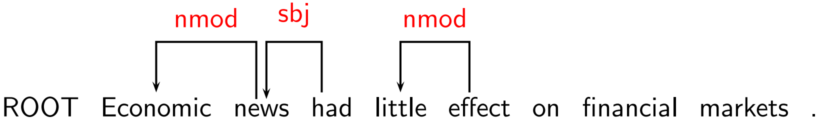
$[\![\text{ROOT} \quad \text{had} \quad \text{effect} \quad \text{on}]\!]_S \quad [\![\text{financial} \quad \text{markets} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

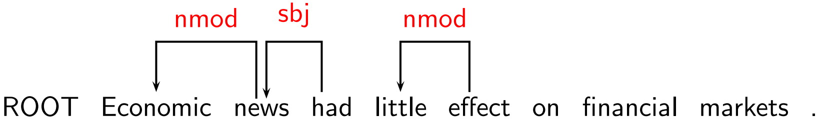
$[\![\text{ROOT} \quad \text{had} \quad \text{effect} \quad \text{on} \quad \text{financial}]\!]_S \quad [\![\text{markets} \quad .]\!]_Q$



Example

$o(c) = \text{Left-Arc}_{nmod}$

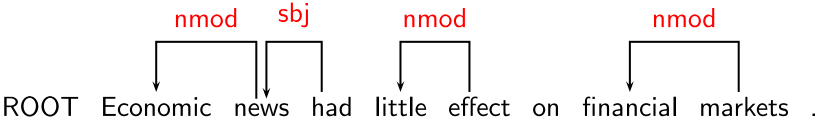
$[\![\text{ROOT had effect on financial markets}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{pc}$

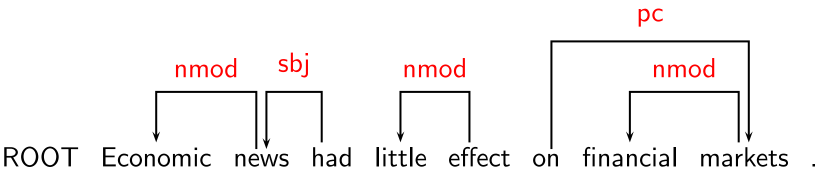
$[\![\text{ROOT} \quad \text{had} \quad \text{effect} \quad \text{on} \quad \text{markets}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{nmod}$

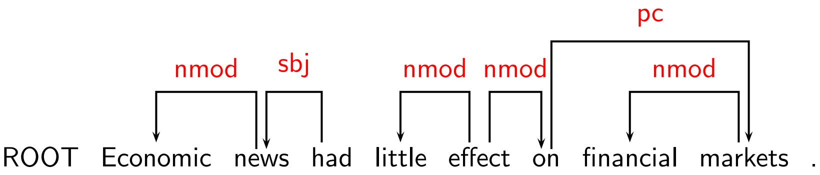
$[\![\text{ROOT} \quad \text{had} \quad \text{effect} \quad \text{on}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{obj}$

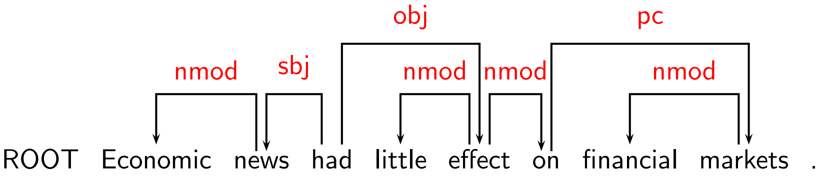
$[\![\text{ROOT} \quad \text{had} \quad \text{effect}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{pred}$

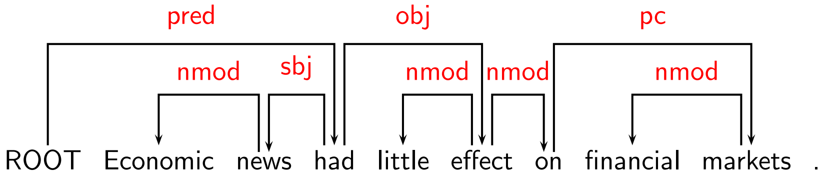
$[\![\text{ROOT} \quad \text{had}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Shift}$

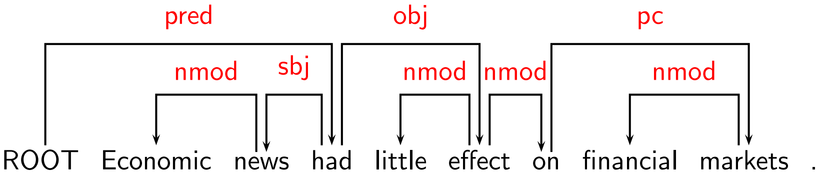
$[\![\text{ROOT}]\!]_S \quad [\![.]\!]_Q$



Example

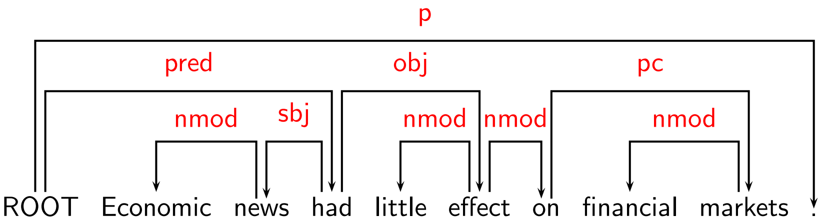
$o(c) = \text{Right-Arc}_p$

$[\![\text{ROOT} \quad .]\!]_S \quad [\![\ ]\!]_Q$



Example

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



Algorithm Analysis

- Given an input sentence of length n, the parser terminates after exactly $2n$ transitions.
- The algorithm is sound and complete for projective dependency trees.
- The algorithm is arguably optimal with respect to
  - robustness (at least one analysis),
  - disambiguation (at most one analysis),
  - efficiency (linear time).
- Accuracy depends on how well we can approximate oracles using machine learning.

Alternative Parsing Algorithms

- Alternative transition systems:
  - Stack-based:
    - Arc-eager shift-reduce parsing [Nivre 2003]
    - Arc-standard shift-reduce parsing [Yamada and Matsumoto 2003]
    - Restricted non-projective parsing [Attardi 2006]
  - List-based:
    - Unrestricted non-projective parsing [Covington 2001, Nivre 2007]
- Alternative search strategies:
  - Greedy search:
    - Single-pass [Nivre et al. 2004]
    - Iterative [Yamada and Matsumoto 2003]
  - Beam search [Johansson and Nugues 2006, Titov and Henderson 2007]

Oracles as Classifiers

- Learning problem in transition-based dependency parsing:
  - Approximate oracle $o(c)$ by classifier $g(c)$
- History-based feature models:
  - Parse history $c = (S, Q, A)$ represented by feature vector $\mathbf{x}(c)$
  - Individual features $\mathbf{x}_i(c)$ defined by properties of words in c, for example:
    - Lexical properties (FORM or LEMMA)
    - Part-of-speech tags (POSTAG)
    - Morphosyntactic features (FEATS)
    - Labels in the partially built dependency tree (DEPREL)

A Typical Feature Model

$$[\ldots \quad w_i \quad w_j \quad w_k \quad]_S \quad [w_m \quad w_{m+1} \quad w_{m+2} \quad \ldots]_Q$$

	w_i	l_j	w_j	r_j	l_k	w_k	r_k	w_m	w_{m+1}	w_{m+2}
FORM			+			+		+		
LEMMA			+			+				
POSTAG	+		+			+		+	+	+
FEATS			+			+				
DEPREL		+		+	+		+			

Training Data

- Training instances have the form $(\mathbf{x}(c), t)$, where
  1. $\mathbf{x}(c)$ is a feature vector representation of a configuration c,
  2. t is the correct transition out of c (i.e., $o(c) = t$).
- Given a dependency treebank, we can sample the oracle function o as follows:
  - For each sentence we reconstruct the transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ for the gold standard dependency tree.
  - For each configuration $c_i (i < m)$, we construct a training instance $(\mathbf{x}(c_i), t_i)$, where $t_i(c_i) = c_{i+1}$.

Learning Algorithms

- Discriminative models for classification:
  - Support vector machines (SVM) [Kudo and Matsumoto 2002, Yamada and Matsumoto 2003, Nivre et al. 2006]
  - Memory-based learning [Nivre et al. 2004, Attardi 2006]
  - Maximum entropy [Cheng et al. 2005, Attardi 2006]
  - Perceptron learning [Ciaramita and Attardi 2007]
- State-of-the-art performance:
  - Deterministic transition-based parsing with SVM classifiers
  - CoNLL Shared Task 2006 and 2007 [Buchholz and Marsi 2006, Nivre et al. 2007]

Summing Up

- The approach so far:
  - Transition systems for constructing dependency trees
  - Deterministic linear-time parsing with oracle
  - Oracles approximated by classifiers trained on treebank data
- However:
  - Limited to projective dependency trees
  - What to do with discontinuous constructions?

Non-Projective Dependency Parsing

What's the Problem?

- Non-projective dependency trees are required for representational adequacy (discontinuity, transparency).
- Non-projective dependency parsing is computationally hard:
  - Exact inference is feasible in polynomial time only with drastic independence assumptions (so-called arc-factored models).
  - Greedy deterministic inference is less efficient than in the projective case ($O(n^2)$ vs. $O(n)$).
- Non-projective dependency parsing is empirically hard:
  - Non-projective dependencies often span longer distances and are hard to learn with data-driven models.

Previous Work

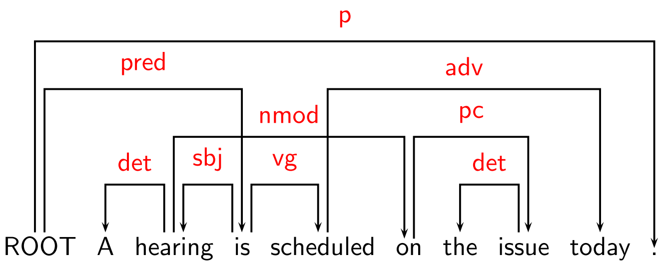
- Algorithms for non-projective dependency parsing:
  - Graph-based parsing using the Chu-Liu-Edmonds algorithm [McDonald et al. 2005b]
  - Transition-based parsing for restricted [Attardi 2006] or arbitrary [Nivre 2007] non-projective structures
- Post-processing of projective dependency trees:
  - Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Corrective modeling [Hall and Novák 2005]
  - Approximate spanning tree parsing [McDonald and Pereira 2006]

A New Idea

- Parsing as the result of two interleaved processes:
  - Sorting the words into a projective order
  - Parsing the sorted words into a projective dependency tree
- Potential advantages:
  - Reduces to projective parsing in the best case
  - Brings elements of discontinuous constructions together

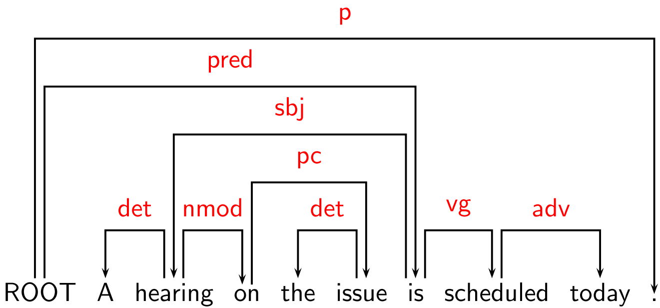
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order.
- Words can always be reordered to make the tree projective.



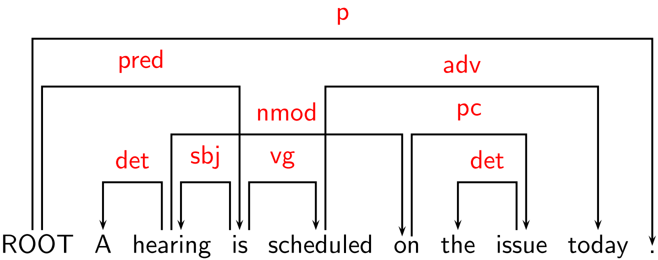
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order.
- Words can always be reordered to make the tree projective.



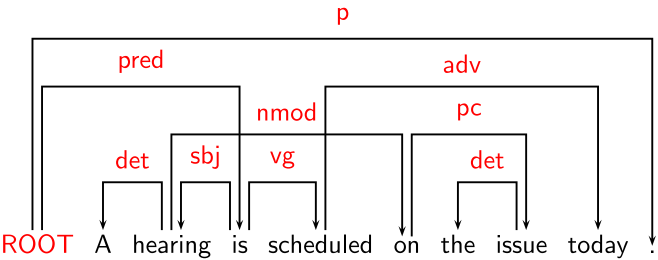
Projective Order

- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



Projective Order

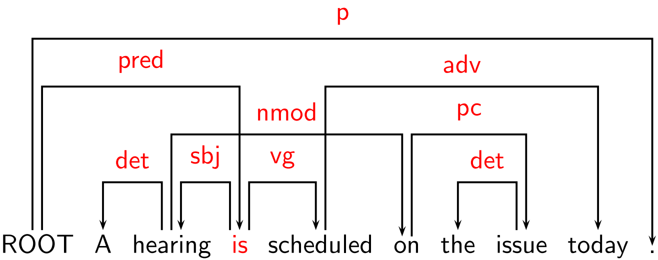
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT

Projective Order

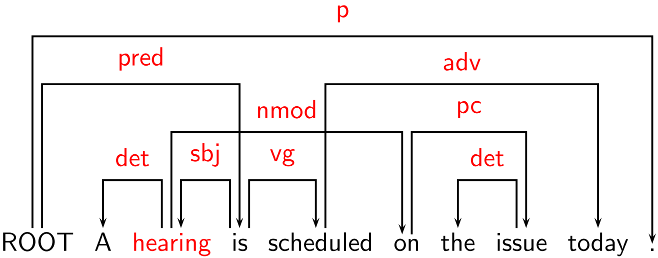
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT

Projective Order

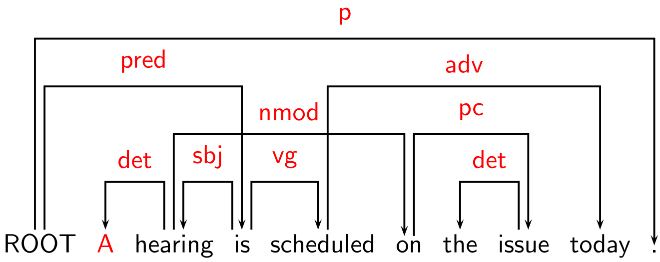
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT

Projective Order

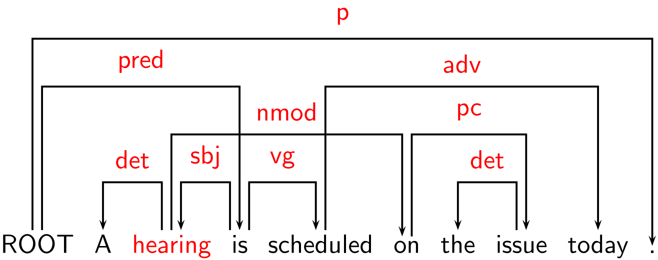
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A

Projective Order

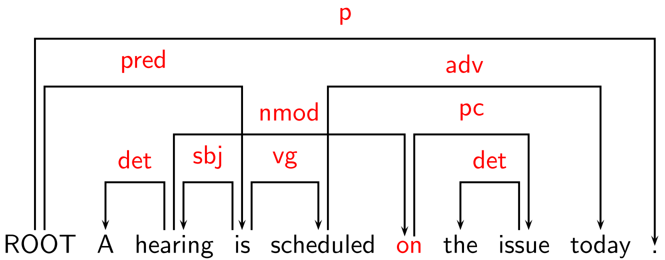
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing

Projective Order

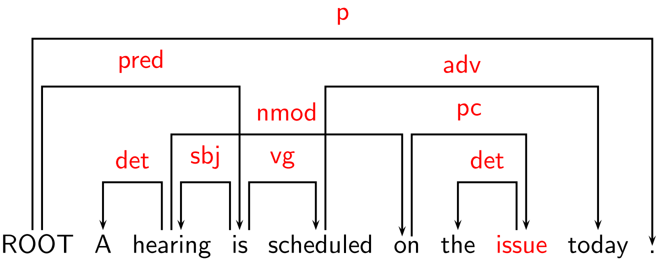
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on

Projective Order

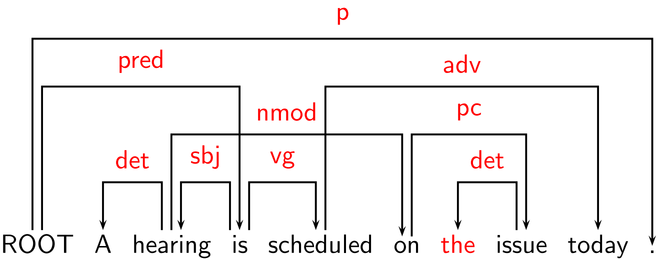
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on

Projective Order

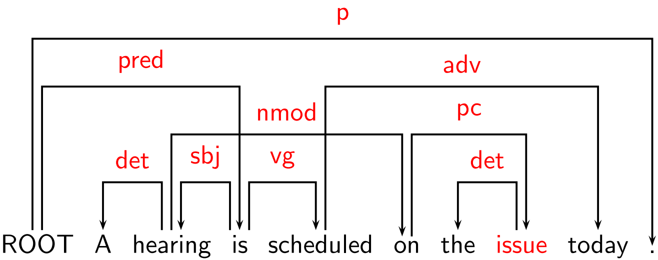
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the

Projective Order

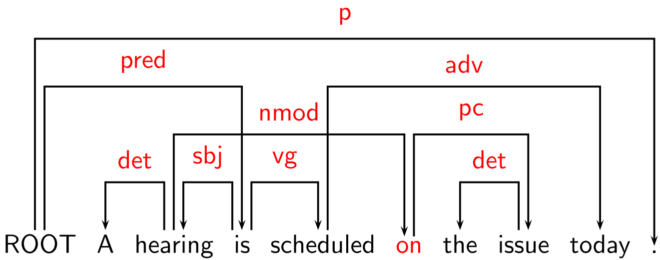
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue

Projective Order

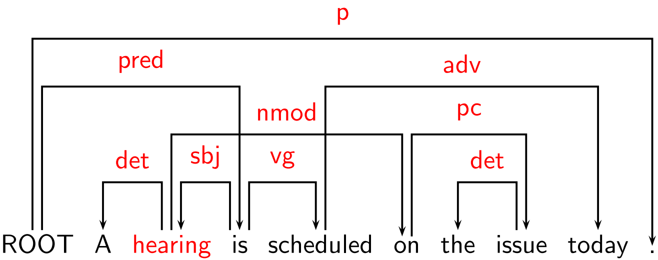
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue

Projective Order

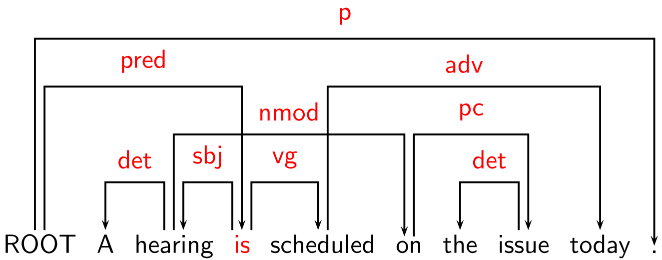
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue

Projective Order

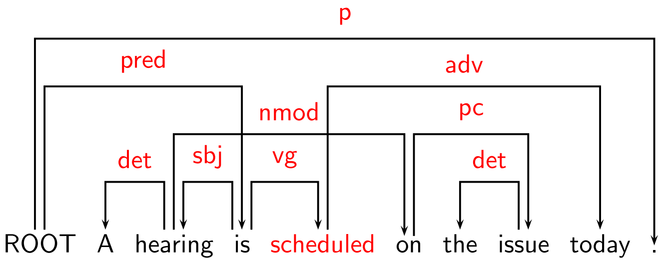
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is

Projective Order

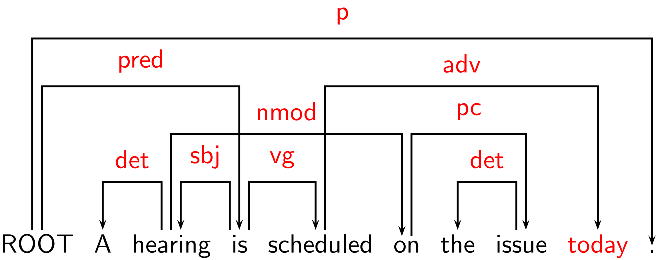
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled

Projective Order

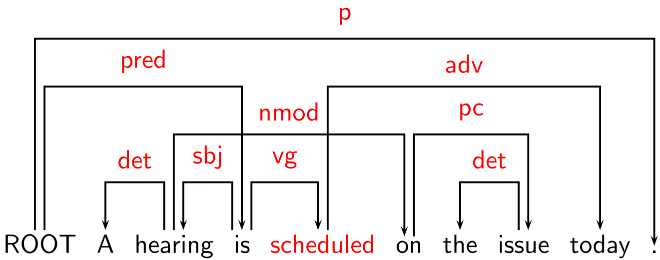
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled today

Projective Order

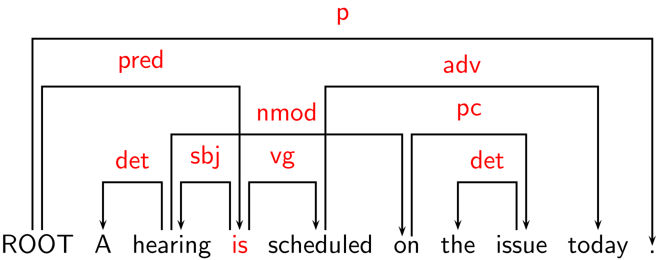
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled today

Projective Order

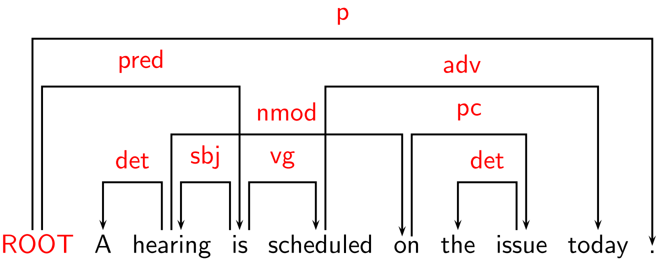
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled today

Projective Order

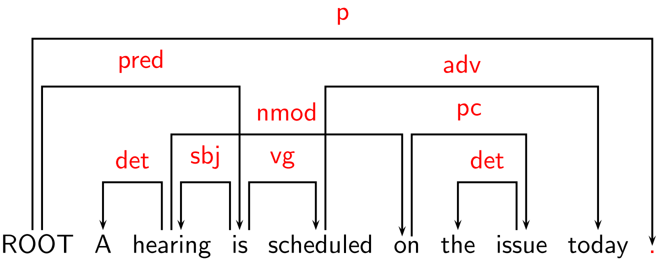
- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled today

Projective Order

- Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$.



ROOT A hearing on the issue is scheduled today .

Sorting into Projective Order

- Basic idea:
  - Combine an algorithm for sorting words according to the projective order $<_p$ with a transition-based algorithm for constructing a projective dependency tree
- Requirements on sorting algorithm:
  - Online algorithm (sorts in a single left-to-right pass)
  - Exchange sort (sorts by swapping elements)
  - Comparison of adjacent elements (cf. parsing algorithm)
- The simplest sorting algorithm:
  - Gnome sort – insertion sort with only adjacent swaps

Sorting into Projective Order

- Basic idea:
  - Combine an algorithm for sorting words according to the projective order $<_p$ with a transition-based algorithm for constructing a projective dependency tree
- Requirements on sorting algorithm:
  - Online algorithm (sorts in a single left-to-right pass)
  - Exchange sort (sorts by swapping elements)
  - Comparison of adjacent elements (cf. parsing algorithm)
- The simplest sorting algorithm:
  - Gnome sort – insertion sort with only adjacent swaps

Transition System: Configurations

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

  $([w_0]_S, [w_1, \ldots, w_n]_Q, \{\,\})$ **NB:** w_0 = ROOT
- Termination:

  $([w_0]_S, [\,]_Q, A)$

Transition System: Transitions

- Swap

$$\frac{([\ldots, w_i, w_j]_S, \quad [\ldots]_Q, \quad A) \quad [i \neq 0, i < j]}{([\ldots, w_j]_S, \quad [w_i, \ldots]_Q, \quad A)}$$

- Left-Arc(l)

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

- Right-Arc(l)

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

- Shift

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

Deterministic Parsing

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

$$
\begin{array}{l}
\text{Parse}(w_1, \ldots, w_n) \\
\quad c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, \{\,\}) \\
\quad \textbf{while } Q_c \neq [\,] \textbf{ or } |S_c| > 1 \\
\qquad t \leftarrow o(c) \\
\qquad c \leftarrow t(c) \\
\quad \textbf{return } G = (\{w_0, w_1, \ldots, w_n\}, A_c)
\end{array}
$$

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT}]\!]_S$ $[\![$A hearing is scheduled on the issue today .$]\!]_Q$

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{A}]\!]_S \quad [\![\text{hearing} \quad \text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .]\!]_Q$

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Left-Arc}_{det}$

$[\![$ROOT A hearing$]\!]_S$ $[\![$is scheduled on the issue today .$]\!]_Q$

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{hearing}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{on} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![$ROOT hearing is$]\!]_S$ $[\![$scheduled on the issue today .$]\!]_Q$

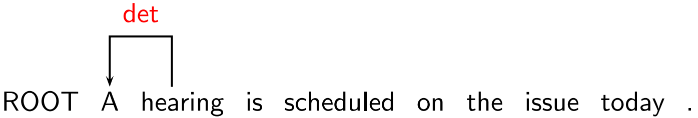
det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![$ROOT hearing is scheduled$]\!]_S$ $[\![$on the issue today .$]\!]_Q$



Example

$o(c) = \text{Swap}$

$[\![$ROOT hearing is scheduled on$]\!]_S$ $[\![$the issue today .$]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Swap}$

$[\![$ROOT hearing is on$]\!]_S$ $[\![$scheduled the issue today .$]\!]_Q$

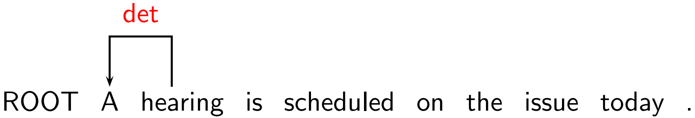
det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{is}]\!]_S \quad [\![\text{scheduled} \quad \text{the} \quad \text{issue} \quad \text{today} \quad .]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![$ROOT hearing on is scheduled$]\!]_S$ $[\![$the issue today .$]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Swap}$

$[\![$ROOT hearing on is scheduled the$]\!]_S$ $[\![$issue today .$]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Swap}$

$[\![\text{ROOT hearing on is the}]\!]_S \; [\![\text{scheduled issue today .}]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT hearing on the}]\!]_S \; [\![\text{is scheduled issue today .}]\!]_Q$

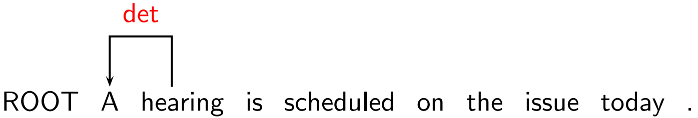
det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{the} \quad \text{is}]\!]_S \quad [\![\text{scheduled} \quad \text{issue} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{the} \quad \text{is} \quad \text{scheduled}]\!]_S \quad [\![\text{issue} \quad \text{today} \quad .]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Swap}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{the} \quad \text{is} \quad \text{scheduled} \quad \text{issue}]\!]_S \quad [\![\text{today} \quad .]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Swap}$

$[\![$ROOT hearing on the is issue$]\!]_S$ $[\![$scheduled today .$]\!]_Q$

det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Left-Arc}_{det}$

$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{the} \quad \text{issue}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{today} \quad .]\!]_Q$

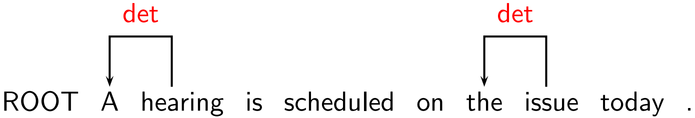
det

ROOT A hearing is scheduled on the issue today .

Example

$o(c) = \text{Right-Arc}_{pc}$

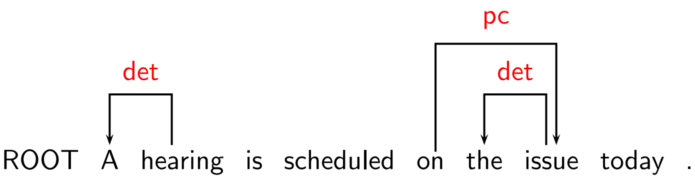
$[\![\text{ROOT} \quad \text{hearing} \quad \text{on} \quad \text{issue}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{nmod}$

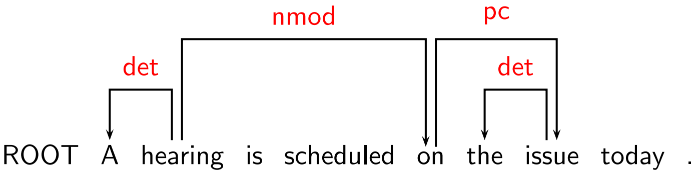
$[\![\text{ROOT} \quad \text{hearing} \quad \text{on}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

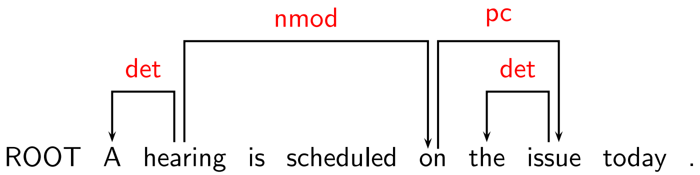
$[\![\text{ROOT} \quad \text{hearing}]\!]_S \quad [\![\text{is} \quad \text{scheduled} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Left-Arc}_{sbj}$

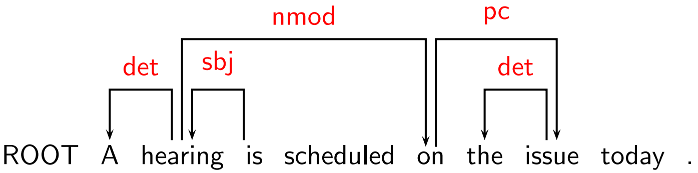
$[\![\text{ROOT} \quad \text{hearing} \quad \text{is}]\!]_S \quad [\![\text{scheduled} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

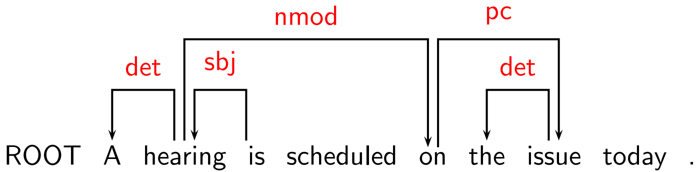
$[\![\text{ROOT} \quad \text{is}]\!]_S \quad [\![\text{scheduled} \quad \text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Shift}$

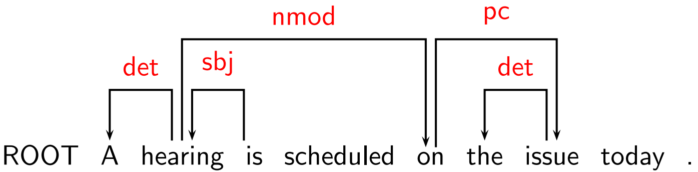
$[\![\text{ROOT} \quad \text{is} \quad \text{scheduled}]\!]_S \quad [\![\text{today} \quad .]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{adv}$

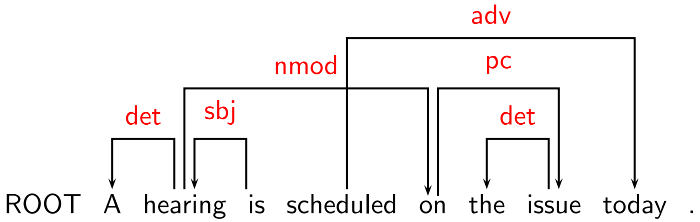
$[\![\text{ROOT} \quad \text{is} \quad \text{scheduled} \quad \text{today}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{vg}$

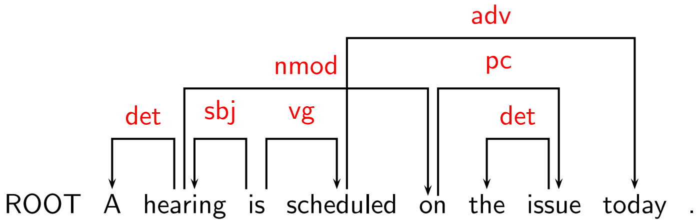
$[\![\text{ROOT} \quad \text{is} \quad \text{scheduled}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Right-Arc}_{pred}$

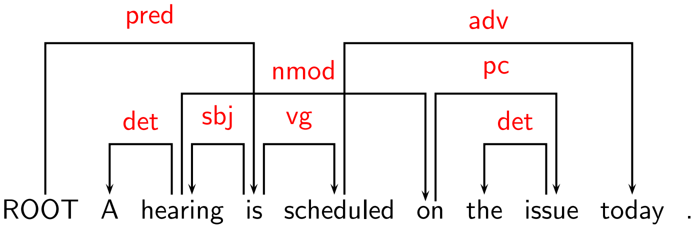
$[\![\text{ROOT} \quad \text{is}]\!]_S \quad [\![.]\!]_Q$



Example

$o(c) = \text{Shift}$

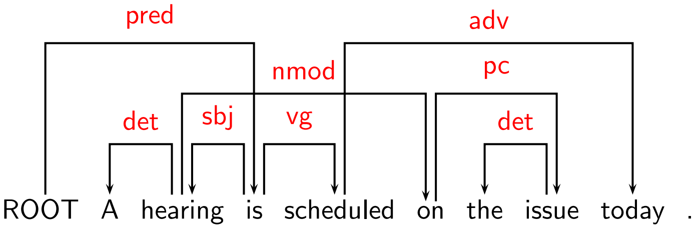
$[\![\text{ROOT}]\!]_S \quad [\![.]\!]_Q$



Example

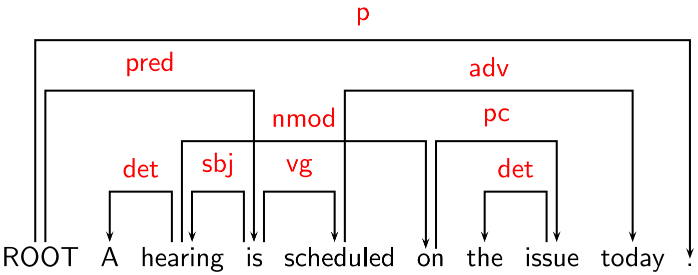
$o(c) = \text{Right-Arc}_p$

$[\![\text{ROOT} \quad .]\!]_S \quad [\![\ ]\!]_Q$



Example

$[\![\text{ROOT}]\!]_S \quad [\![\;]\!]_Q$



Algorithm Analysis

- Time complexity of parsing:
  - $O(n^2)$ in the worst case ($\frac{n(n-1)}{2}$ swaps)
  - $O(n)$ in the best case (0 swaps)
  - Average case $\approx$ best case?
- Conjecture:
  - Sound and complete for non-projective dependency trees
- Crucial question:
  - Can we train classifiers to do sorting as well as parsing?

Experimental Evaluation

- Data from the CoNLL shared tasks [Buchholz and Marsi 2006, Nivre et al. 2007]:
  - Prague Arabic Dependency Treebank (2007) (2.9k, 10.1%)
  - Prague Dependency Treebank (2007) (25.4k, 23.2%)
  - Slovene Dependency Treebank (2006) (1.5k, 22.2%)
  - Metu-Sabancı Turkish Treebank (2007) (5.6k, 33.3%)
- Classifiers:
  - Support vector machines with polynomial kernel (degree 2)
  - Feature models optimized on development set
- Evaluation metric:
  - Labeled attachment score (LAS): Percentage of words that are assigned the correct head and dependency label

Preliminary Results

Parser	Arabic	Czech	Slovene	Turkish	Average
Projective					
Pseudo-projective					
Non-projective					
MaltParser 06/07					
Top score 06/07					

Preliminary Results

Parser	Arabic	Czech	Slovene	Turkish	Average
Projective	75.8	74.0	73.3	79.0	75.5
Pseudo-projective					
Non-projective					
MaltParser 06/07					
Top score 06/07					

Preliminary Results

Parser	Arabic	Czech	Slovene	Turkish	Average
Projective	75.8	74.0	73.3	79.0	75.5
Pseudo-projective	76.1	78.2	74.6	79.6	77.1
Non-projective					
MaltParser 06/07					
Top score 06/07					

Preliminary Results

Parser	Arabic	Czech	Slovene	Turkish	Average
Projective	75.8	74.0	73.3	79.0	75.5
Pseudo-projective	76.1	78.2	74.6	79.6	77.1
Non-projective	76.2	79.2	75.4	79.6	77.6
MaltParser 06/07					
Top score 06/07					

Preliminary Results

Parser	Arabic	Czech	Slovene	Turkish	Average
Projective	75.8	74.0	73.3	79.0	75.5
Pseudo-projective	76.1	78.2	74.6	79.6	77.1
Non-projective	76.2	79.2	75.4	79.6	77.6
MaltParser 06/07	74.8	77.2	70.3	79.2	75.4
Top score 06/07	76.5	80.2	73.4	79.8	77.5

Conclusion

- Transition-based dependency parsing:
  - Efficient thanks to greedy, deterministic search
  - Accurate thanks to powerful discriminative classification
- Novel approach to non-projective dependency parsing:
  - Interleaved sorting and parsing
  - Efficiency maintained
  - Promising empirical results

Acknowledgments

- My students and co-developers of MaltParser:
  - Johan Hall and Jens Nilsson
- Useful comments from:
  - John Carroll, Carlos Gómez-Rodríguez, Marco Kuhlmann, Ryan McDonald, Paola Merlo, Jamie Henderson, Peter Ljunglöf, Bengt Nordström, Kenji Sagae, David Weir and three anonymous reviewers

- I. Aduriz, M. J. Aranzabe, J. M. Arriola, A. Atutxa, A. Díaz de Ilarraza, A. Garmendia, and M. Oronoz. 2003. Construction of a Basque dependency treebank. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 201–204.
- Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.
- Igor Boguslavsky, Svetlana Grigorieva, Nikolai Grigoriev, Leonid Kreidlin, and Nadezhda Frid. 2000. Dependency treebank for Russian: Concept, tools, types of information. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 987–991.
- Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.
- Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005. Machine learning-based dependency analyzer for Chinese. In *Proceedings of International Conference on Chinese Computing (ICCC)*, pages 66–73.
- Massimiliano Ciaramita and Giuseppe Attardi. 2007. Dependency parsing with second-order feature maps and annotated semantic information. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 133–143, June.
- Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.
- Sašo Džeroski, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdenek Žabokrtsky, and Andreja Žele. 2006. Towards a Slovene dependency treebank. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*.
- Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.
- Jan Hajič, Barbora Vidova Hladka, Jarmila Panevová, Eva Hajičová, Petr Sgall, and Petr Pajas. 2001. Prague Dependency Treebank 1.0. LDC, 2001T10.

- Keith Hall and Vaclav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pages 42–52.
- Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 206–210.
- Matthias Trautner Kromann. 2003. The Danish Dependency Treebank and the DTAG treebank tool. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 217–220.
- Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pages 63–69.
- Mohamed Maamouri and Ann Bies. 2004. Developing an Arabic treebank: Methods, guidelines, procedures, and tools. In *Proceedings of the Workshop on Computational Approaches to Arabic Script-Based Languages*, pages 2–9.
- Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88.
- Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530.
- Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.
- Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning*, pages 49–56.

- Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.
- Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.
- Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.
- Joakim Nivre. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 396–403.
- Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer.
- P. Prokopidis, E. Desypri, M. Koutsombogera, H. Papageorgiou, and S. Piperidis. 2005. Theoretical and practical issues in the construction of a Greek dependency treebank. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 149–160.
- Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 144–155.
- Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.