Advanced Transition-Based Parsing Techniques

Joakim Nivre

Uppsala University
Linguistics and Philology

Based on previous tutorials with Ryan McDonald

Overall Plan

1. Basic notions of dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Advanced graph-based parsing techniques
4. Advanced transition-based parsing techniques
5. Neural network techniques in dependency parsing
6. Multilingual parsing from raw text to universal dependencies

Plan for this Lecture

- Improved learning and inference
  - Beam search and structured prediction
  - Easy-first parsing
  - Dynamic oracles
- Non-projective parsing using online reordering
- Joint morphological and syntactic analysis

Transition-Based Parsing Trade-Off

- 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

Beam Search

- Maintain the k best hypotheses [Johansson and Nugues 2006]:

  Parse($w_1, \ldots, w_n$)
  Beam $\leftarrow \{([\ ]_S, [0, 1, \ldots, n]_B, \{\ \})\}$
  while $\exists c \in$ Beam $[B_c \neq [\ ]]$
    foreach $c \in$ Beam
      foreach t
        Add($t(c)$, NewBeam)
    Beam $\leftarrow$ Top(k, NewBeam)
  return $G = (\{0, 1, \ldots, n\}, A_{\text{Top}(1,\,\text{Beam})})$

- Note:
  - $\text{Score}(c_0, \ldots, c_m) = \sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{f}(c_{i-1}, t_i)$
  - Simple combination of locally normalized classifier scores
  - Marginal gains in accuracy

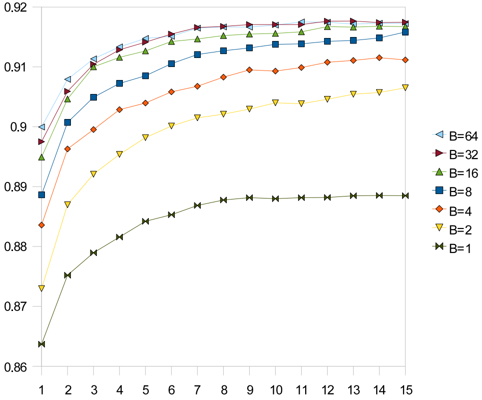
Structured Prediction

- Parsing as structured prediction [Zhang and Clark 2008]:
  - Minimize loss over entire transition sequence
  - Use beam search to find highest-scoring sequence
- Factored feature representations:

$$\mathbf{f}(c_0, \ldots, c_m) = \sum_{i=1}^{m} \mathbf{f}(c_{i-1}, t_i)$$

- Online learning from oracle transition sequences:
  - Structured perceptron [Collins 2002]
  - Early update [Collins and Roark 2004]
  - Max-violation update [Huang et al. 2012]

Beam Size and Training Iterations

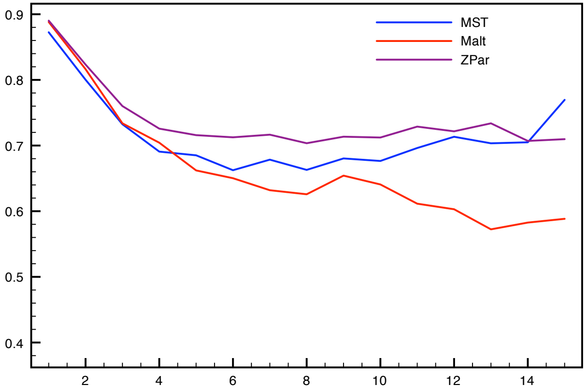


[Zhang and Clark 2008]

The Best of Two Worlds?

- Like graph-based dependency parsing (MSTParser):
  - Global learning – minimize loss over entire sentence
  - Non-greedy search – accuracy increases with beam size
- Like (old school) transition-based parsing (MaltParser):
  - Highly efficient – complexity still linear for fixed beam size
  - Rich features – no constraints from parsing algorithm

Precision by Dependency Length



[Zhang and Nivre 2012]

Even Richer Feature Models

	ZPar	Malt
Baseline	92.18	89.37
+distance	+0.07	–0.14
+valency	+0.24	0.00
+unigrams	+0.40	–0.29
+third-order	+0.18	0.00
+label set	+0.07	+0.06
Extended	93.14	89.00

[Zhang and Nivre 2011, Zhang and Nivre 2012]

- Adding graph-based features may require special techniques
  [Zhang and Clark 2008, Bohnet and Kuhn 2012]

The Need for Speed

- Beam search helps but slows down the parser
- What can we do to maintain the highest speed?
  - Easy-first parsing – give up left-to-right incremental search
  - Dynamic oracles – learn how to recover from errors
- These two ideas can be combined

Easy-First Non-Directional Parsing

- Process dependencies from easy to hard (not left to right) and from local to global (bottom up) [Goldberg and Elhadad 2010]

Configuration: (L, A) $[L = \text{List}, A = \text{Arcs}]$

Initial: $([0, 1, \ldots, n], \{ \})$

Terminal: $([0], A)$

Attach-Right(i, k):
$([v_1, \ldots, v_m], A) \Rightarrow ([v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m], A \cup \{(v_{i+1}, v_i, k)\})$

Attach-Left(i, k):
$([v_1, \ldots, v_m], A) \Rightarrow ([v_1, \ldots, v_i, v_{i+2}, \ldots, v_m], A \cup \{(v_i, v_{i+1}, k)\})$

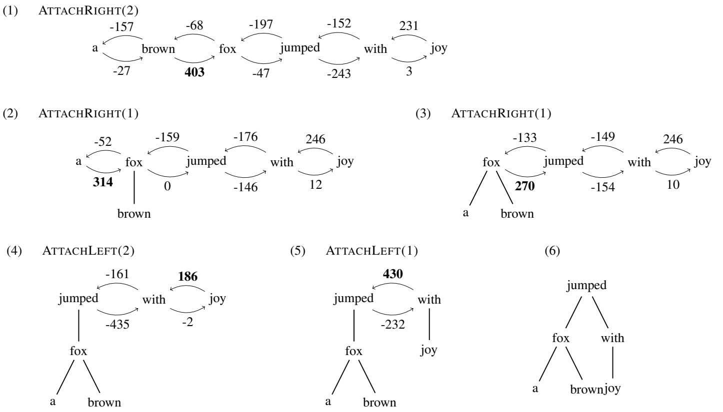
Parsing Algorithm

- Given an oracle o that selects the highest-confidence transition $o(c)$, parsing is deterministic:

Parse($w_1, \ldots, w_n$)
    $c \leftarrow ([0, 1, \ldots, n], \{\ \})$
    while length(L_c) > 1
        $t \leftarrow o(c)$
        $c \leftarrow t(c)$
    return $G = (\{0, 1, \ldots, n\}, A_c)$

- Number of possible transitions grows with sentence length
- Parsing in $O(n \log n)$ time with priority heap

Parsing Example



[Goldberg and Elhadad 2010]

Oracles Revisited

- How do we train the easy-first parser?
- Recall our training procedure for greedy parsers:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set $D = \{(c, t) \mid o(c) = t\}$
  - Maximize accuracy of local predictions $o(c) = t$
- Presupposes a unique optimal transition for each configuration
  - Does not make sense for the easy-first parser
  - Turns out to be a bad idea in general

Online Learning with a Conventional Oracle

Learn($\{T_1, \ldots, T_N\}$)

$\mathbf{w} \leftarrow 0.0$
for i in $1..K$
    for j in $1..N$
        $c \leftarrow ([\,], [0, 1, \ldots, n_j], \{\,\})$
        while $B_c \neq [\,]$
            $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            $t_o \leftarrow o(c, T_i)$
            if $t^* \neq t_o$
                $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
            $c \leftarrow t_o(c)$
return w

Online Learning with a Conventional Oracle

Learn($\{T_1, \ldots, T_N\}$)

$\mathbf{w} \leftarrow 0.0$
for i in $1..K$
    for j in $1..N$
        $c \leftarrow ([\,], [0, 1, \ldots, n_j], \{\,\})$
        while $B_c \neq [\,]$
            $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            $t_o \leftarrow o(c, T_i)$
            if $t^* \neq t_o$
                $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
            $c \leftarrow t_o(c)$
return w

- Oracle $o(c, T_i)$ returns the optimal transition for c and T_i

Conventional Oracle for Arc-Eager Parsing

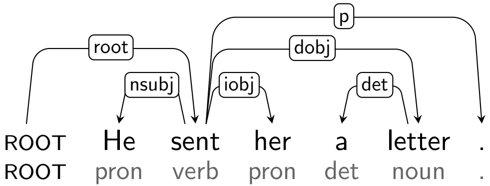
$$o(c, T) = \begin{cases} \text{Left-Arc} & \text{if } \text{top}(S_c) \leftarrow \text{first}(B_c) \text{ in } T \\ \text{Right-Arc} & \text{if } \text{top}(S_c) \rightarrow \text{first}(B_c) \text{ in } T \\ \text{Reduce} & \text{if } \exists v < \text{top}(S_c) : v \leftrightarrow \text{first}(B_c) \text{ in } T \\ \text{Shift} & \text{otherwise} \end{cases}$$

- Correct:
  - Derives T in a configuration sequence $C_{o,T} = c_0, \ldots, c_m$
- Problems:
  - Deterministic: Ignores other derivations of T
  - Incomplete: Valid only for configurations in $C_{o,T}$

Oracle Parse

Transitions:

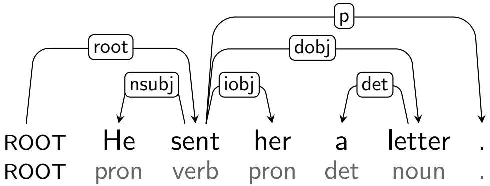
Stack	**Buffer**	**Arcs**
[ ]	[ROOT, He, sent, her, a, letter, .]	



Oracle Parse

Transitions: SH

Stack	**Buffer**	**Arcs**
[ROOT]	[He, sent, her, a, letter, .]	



Oracle Parse

Transitions: SH-RA

Stack

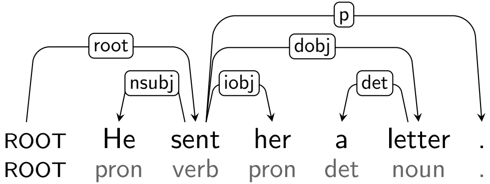
[ROOT, He]

Buffer

[sent, her, a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent



Oracle Parse

Transitions: SH-RA-LA

Stack

[ROOT]

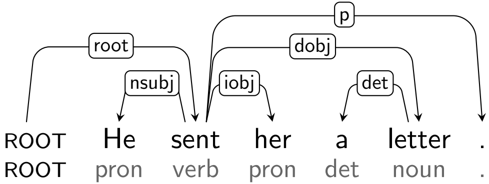
Buffer

[sent, her, a, letter, .]

Arcs

$\text{ROOT} \xrightarrow{\text{root}} \text{sent}$

$\text{He} \xleftarrow{\text{sbj}} \text{sent}$



Oracle Parse

Transitions: SH-RA-LA-SH

Stack

[ROOT, sent]

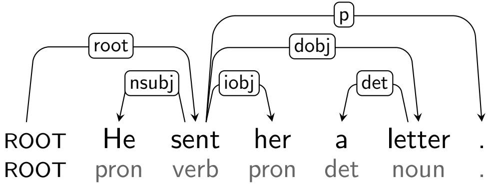
Buffer

[her, a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent



Oracle Parse

Transitions: SH-RA-LA-SH-RA

Stack

[ROOT, sent, her]

Buffer

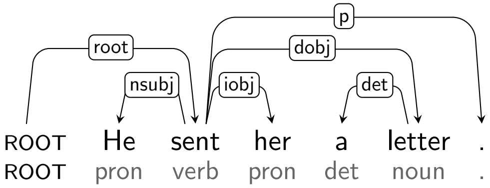
[a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH

Stack

[ROOT, sent, her, a]

Buffer

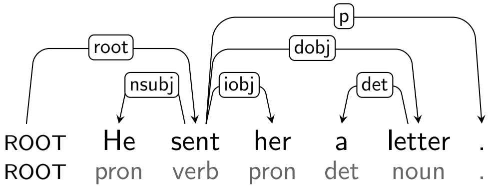
[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH-LA

Stack

[ROOT, sent, her]

Buffer

[letter, .]

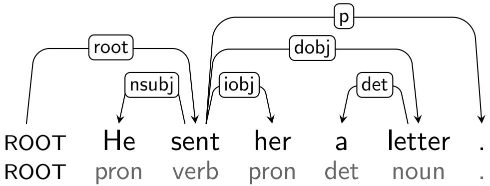
Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH-LA-RE

Stack

[ROOT, sent]

Buffer

[letter, .]

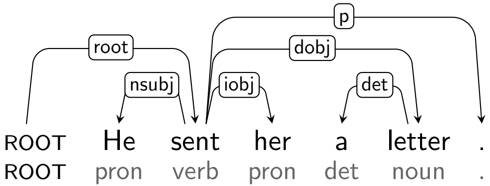
Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH-LA-RE-RA

Stack

[ROOT, sent, letter]

Buffer

[.]

Arcs

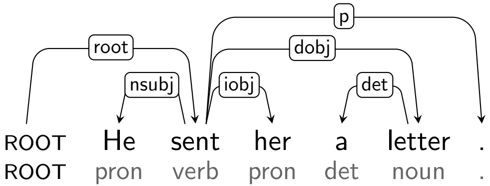
ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH-LA-RE-RA-RE

Stack

[ROOT, sent]

Buffer

[.]

Arcs

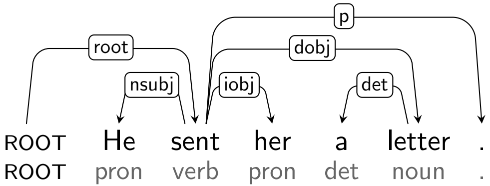
ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Oracle Parse

Transitions: SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

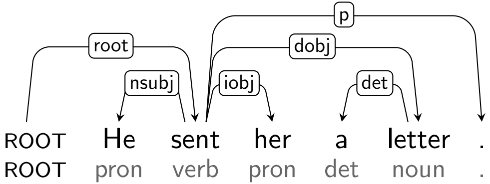
He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

sent $\xrightarrow{\text{p}}$.



Non-Determinisim

Transitions: SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA

Stack

[ROOT, sent, her]

Buffer

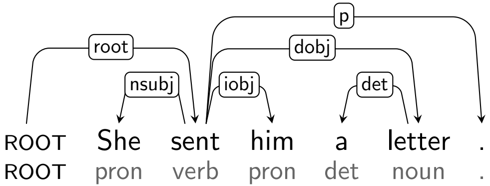
[a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions: SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA-RE

Stack

[ROOT, sent]

Buffer

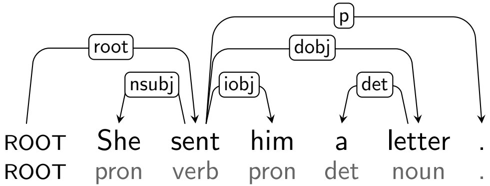
[a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA-RE-SH

Stack

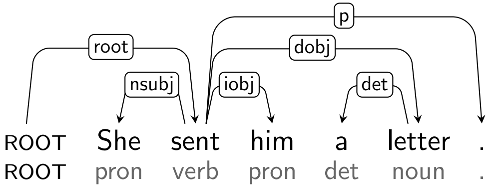
[ROOT, sent, a]

Buffer

[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{sbj}}$ sent
sent $\xrightarrow{\text{iobj}}$ her



Non-Determinisim

Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA-RE-SH-LA

Stack

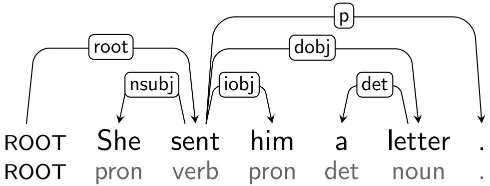
[ROOT, sent]

Buffer

[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{sbj}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter



Non-Determinisim

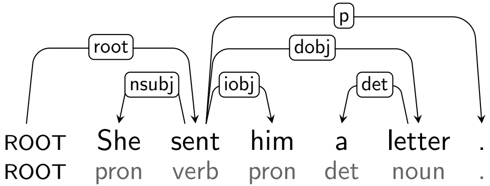
Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA-RE-SH-LA-RA

Stack	**Buffer**
[ROOT, sent, letter]	[.]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{sbj}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter
sent $\xrightarrow{\text{dobj}}$ letter



Non-Determinisim

Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-RA-LA-SH-RA-RE-SH-LA-RA-RE

Stack

[ROOT, sent]

Buffer

[.]

Arcs

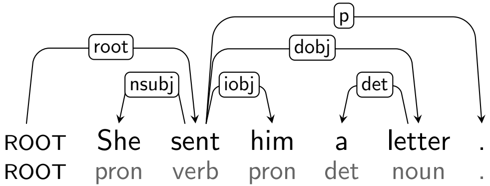
ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

sent $\xrightarrow{\text{iobj}}$ her

a $\xleftarrow{\text{det}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



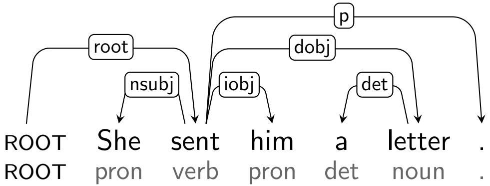
Non-Determinisim

Transitions:
SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
SH-RA-LA-SH-RA-RE-SH-LA-RA-RE-RA

Stack
[ROOT, sent, .]

Buffer
[ ]

Arcs
ROOT $\xrightarrow{\text{root}}$ sent
He $\xleftarrow{\text{sbj}}$ sent
sent $\xrightarrow{\text{iobj}}$ her
a $\xleftarrow{\text{det}}$ letter
sent $\xrightarrow{\text{dobj}}$ letter
sent $\xrightarrow{\text{p}}$.



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH

Stack

[ROOT, sent]

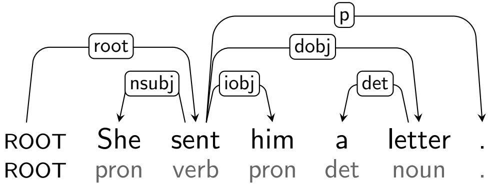
Buffer

[her, a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH

Stack

[ROOT, sent, her]

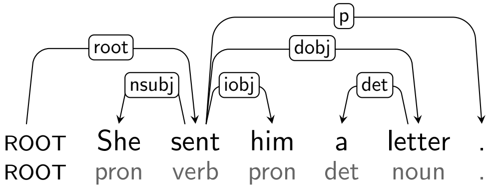
Buffer

[a, letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH-SH

Stack

[ROOT, sent, her, a]

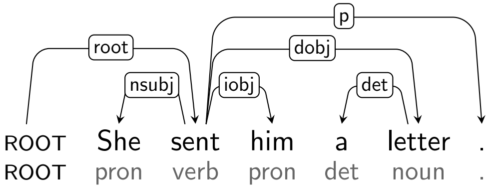
Buffer

[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH-SH-LA

Stack

[ROOT, sent, her]

Buffer

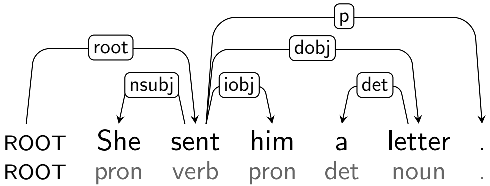
[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH-SH-LA-SH

Stack

[ROOT, sent, her, letter]

Buffer

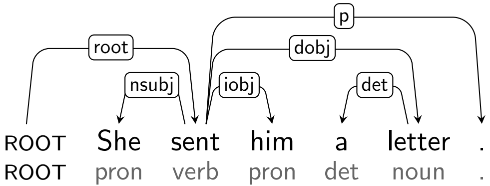
[.]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH-SH-LA-SH-SH [3/6]

Stack

[ROOT, sent, letter, .]

Buffer

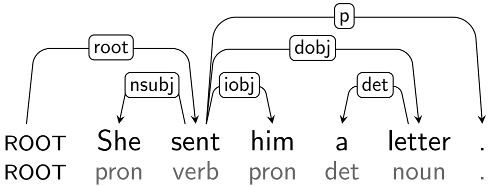
[ ]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

	SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA	
Transitions:	SH-RA-LA-SH-SH-SH-LA-SH-SH	[3/6]
	SH-RA-LA-SH-SH-SH-LA	

Stack

[ROOT, sent, her]

Buffer

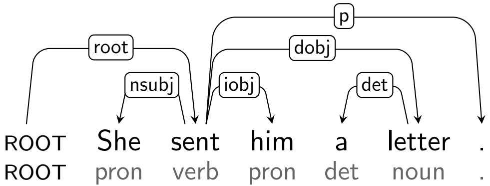
[letter, .]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter



Non-Optimality

	SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA
Transitions:	SH-RA-LA-SH-SH-SH-LA-SH-SH [3/6]
	SH-RA-LA-SH-SH-SH-LA-LA

Stack

[ROOT, sent]

Buffer

[letter, .]

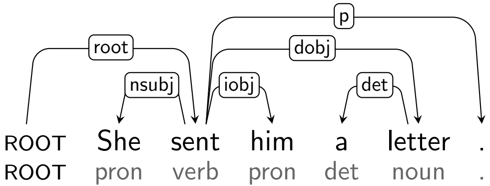
Arcs

ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter



Non-Optimality

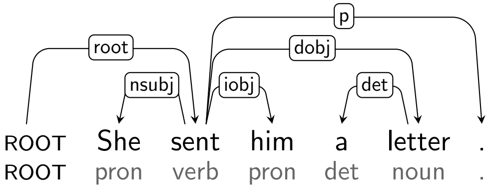
Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-RA-LA-SH-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA

Stack	**Buffer**	**Arcs**
[ROOT, sent, letter]	[.]	ROOT $\xrightarrow{\text{root}}$ sent
		He $\xleftarrow{\text{sbj}}$ sent
		a $\xleftarrow{\text{det}}$ letter
		her $\xleftarrow{?}$ letter
		sent $\xrightarrow{\text{dobj}}$ letter



Non-Optimality

Transitions:

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

SH-RA-LA-SH-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA-RE

Stack

[ROOT, sent]

Buffer

[.]

Arcs

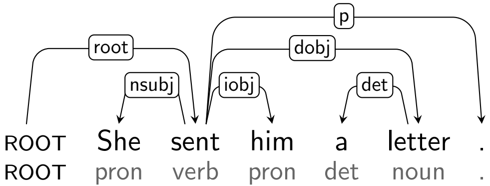
ROOT $\xrightarrow{\text{root}}$ sent

He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{?}$ letter

sent $\xrightarrow{\text{dobj}}$ letter



Non-Optimality

SH-RA-LA-SH-RA-SH-LA-RE-RA-RE-RA

Transitions: SH-RA-LA-SH-SH-SH-LA-SH-SH [3/6]

SH-RA-LA-SH-SH-SH-LA-LA-RA-RE-RA [5/6]

Stack

[ROOT, sent, .]

Buffer

[ ]

Arcs

ROOT $\xrightarrow{\text{root}}$ sent

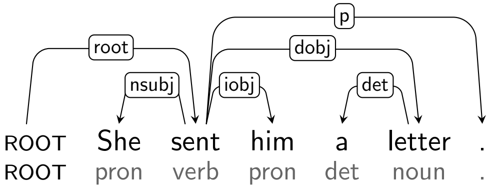
He $\xleftarrow{\text{sbj}}$ sent

a $\xleftarrow{\text{det}}$ letter

her $\xleftarrow{\text{?}}$ letter

sent $\xrightarrow{\text{dobj}}$ letter

sent $\xrightarrow{\text{p}}$.



Dynamic Oracles

- Optimality:
  - A transition is optimal if the best tree remains reachable
  - Best tree = $\text{argmin}_{T'} \mathcal{L}(T, T')$
- Oracle:
  - Boolean function $o(c, t, T) =$ **true** if t is optimal for c and T
  - Non-deterministic: More than one transition can be optimal
  - Complete: Correct for all configurations
- New problem:
  - How do we know which trees are reachable?

Reachability for Arcs and Trees

- Arc reachability:
  - An arc $w_i \rightarrow w_j$ is reachable in c iff $w_i \rightarrow w_j \in A_c$, or $w_i \in S_c \cup B_c$ and $w_j \in B_c$ (same for $w_i \leftarrow w_j$)
- Tree reachability:
  - A (projective) tree T is reachable in c iff every arc in T is reachable in c
- Arc-decomposable systems [Goldberg and Nivre 2013]:
  - Tree reachability reduces to arc reachability
  - Holds for some transition systems but not all
    - Arc-eager and easy-first are arc-decomposable
    - Arc-standard is not decomposable

Oracles for Arc-Decomposable Systems

$$o(c, t, T) = \begin{cases} \textbf{true} & \text{if } [\mathcal{R}(c) - \mathcal{R}(t(c))] \cap T = \emptyset \\ \textbf{false} & \text{otherwise} \end{cases}$$

$$\text{where } \mathcal{R}(c) \equiv \{a \mid a \text{ is an arc reachable in } c\}$$

Arc-Eager

$$o(c, \text{LA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \leftrightarrow w \in T \text{ (except } s \leftarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RA}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \text{ (except } s \rightarrow b) \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{RE}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in B_c : s \rightarrow w \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

$$o(c, \text{SH}, T) = \begin{cases} \textbf{false} & \text{if } \exists w \in S_c : w \leftrightarrow b \in T \\ \textbf{true} & \text{otherwise} \end{cases}$$

Notation: s = node on top of the stack S
b = first node in the buffer B

Online Learning with a Dynamic Oracle

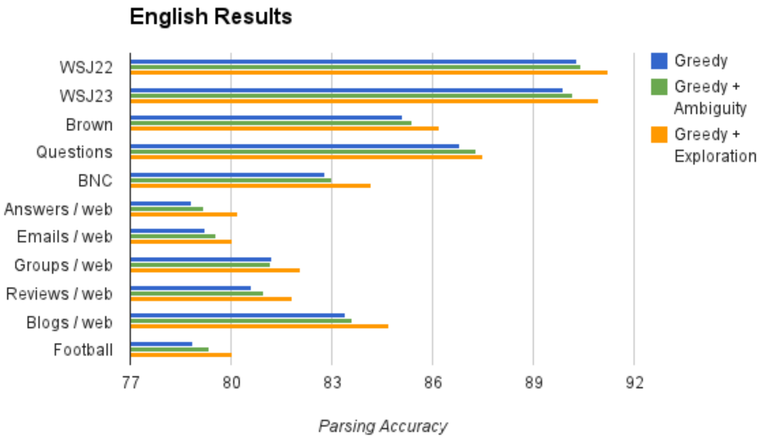
Learn($\{T_1, \ldots, T_N\}$)

$\mathbf{w} \leftarrow 0.0$
for i in $1..K$
    for j in $1..N$
        $c \leftarrow ([\,]_S, [w_1, \ldots, w_{n_j}]_B, \{\,\})$
        while $B_c \neq [\,]$
            $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            $t_o \leftarrow \text{argmax}_{t \in \{t | o(c,t,T_i)\}}\, \mathbf{w} \cdot \mathbf{f}(c, t)$
            if $t^* \neq t_o$
                $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
            $c \leftarrow \text{choice}(t_o(c), t^*(c))$
return $\mathbf{w}$

Online Learning with a Dynamic Oracle

Learn($\{T_1, \ldots, T_N\}$)
    $\mathbf{w} \leftarrow 0.0$
    for i in $1..K$
        for j in $1..N$
            $c \leftarrow ([\,]_S, [w_1, \ldots, w_{n_j}]_B, \{\,\})$
            while $B_c \neq [\,]$
                $t^* \leftarrow \text{argmax}_t\, \mathbf{w} \cdot \mathbf{f}(c, t)$
                $t_o \leftarrow \text{argmax}_{t \in \{t \mid o(c,t,T_i)\}}\, \mathbf{w} \cdot \mathbf{f}(c, t)$
                if $t^* \neq t_o$
                    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(c, t_o) - \mathbf{f}(c, t^*)$
                $c \leftarrow \text{choice}(t_o(c), t^*(c))$
    return w

- Ambiguity: use model score to break ties
- Exploration: follow model prediction even if not optimal



[Goldberg and Nivre 2012]

Ambiguity and Exploration

- Lessons from dynamic oracles:
  - Do not hide spurious ambiguity from the parser – exploit it
  - Let the parser explore the consequences of its own mistakes
- Related work:
  - Bootstrapping [Choi and Palmer 2011]
  - Selectional branching [Choi and McCallum 2013]
  - Non-monotonic parsing [Honnibal et al. 2013]
  - Dynamic parsing strategy [Sartorio et al. 2013]

Non-Projective Parsing

- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: $n(n-1)$ arcs to consider – $O(n^2)$
  - Projective: at most $2(n-1)$ arcs to consider – $O(n)$
- Also harder to construct dynamic oracles
  - Conjecture: arc-decomposability presupposes projectivity

Previous Approaches

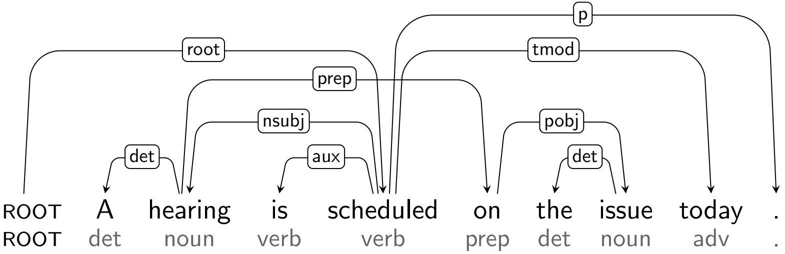
- Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Preprocess training data, post-process parser output
  - Approximate encoding with incomplete coverage
  - Relatively high precision but low recall
- Extended arc transitions [Attardi 2006]
  - Transitions that add arcs between non-adjacent subtrees
  - Upper bound on arc degree (limited to local relations)
  - Exact dynamic programming algorithm [Cohen et al. 2011]
- List-based algorithms [Covington 2001, Nivre 2007]
  - Consider all word pairs instead of adjacent subtrees
  - Increases parsing complexity (and training time)
  - Improved accuracy and efficiency by adding "projective transitions" [Choi and Palmer 2011]

Novel Approaches

- Online reordering [Nivre 2009, Nivre et al. 2009]:
  - Reorder words during parsing to make tree projective
  - Add a special transition for swapping adjacent words
  - Quadratic time in the worst case but linear in the best case
- Multiplanar parsing [Gómez-Rodríguez and Nivre 2010]:
  - Factor dependency trees into k planes without crossing arcs
  - Use k stacks to parse each plane separately
  - Linear time parsing with constant k

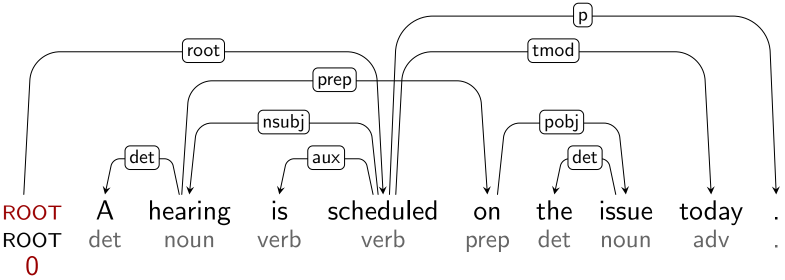
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



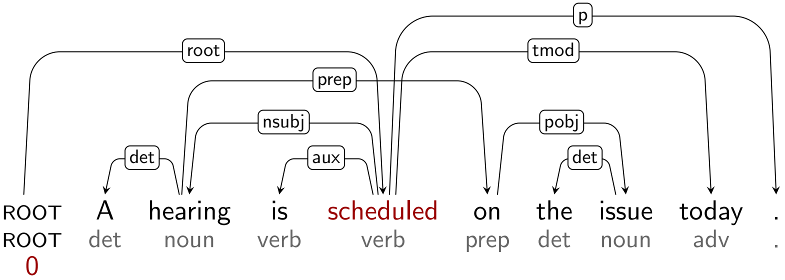
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



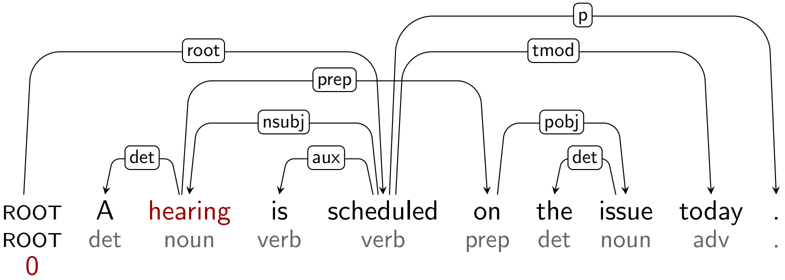
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



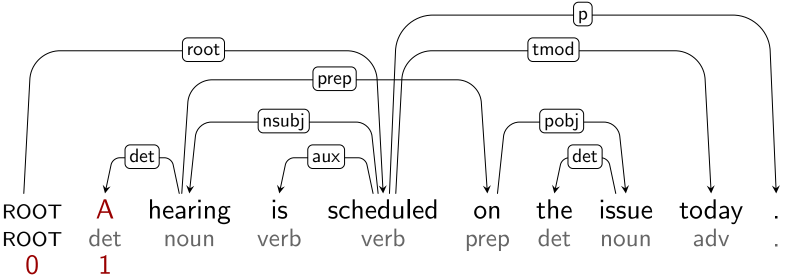
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



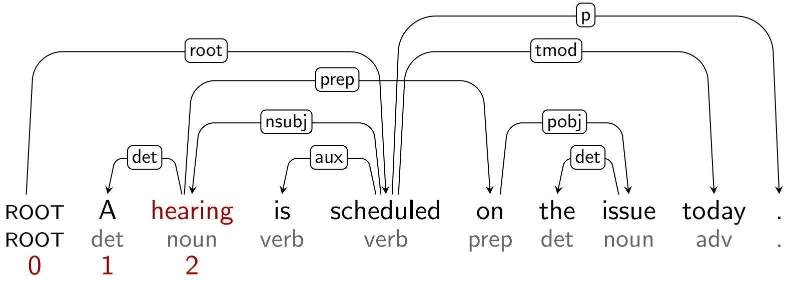
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



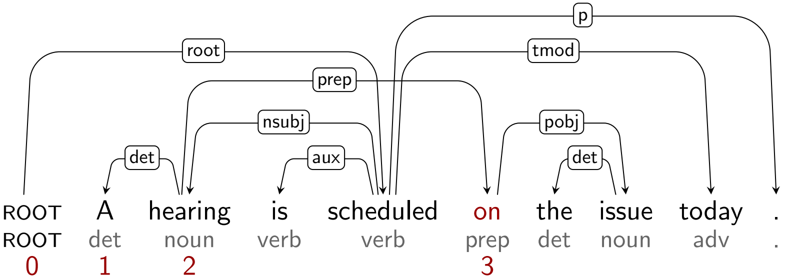
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



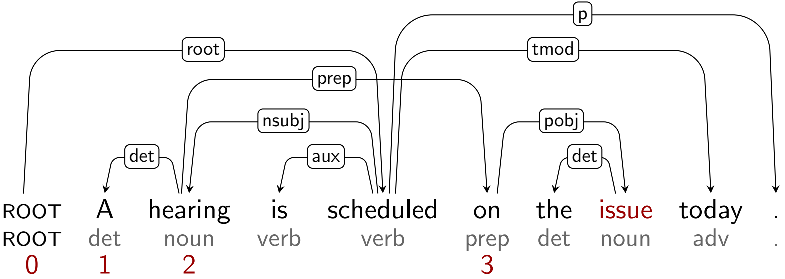
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



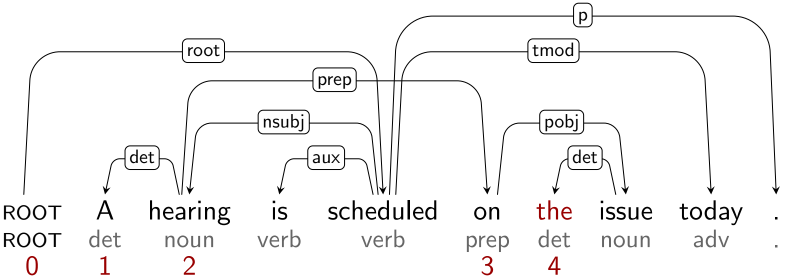
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



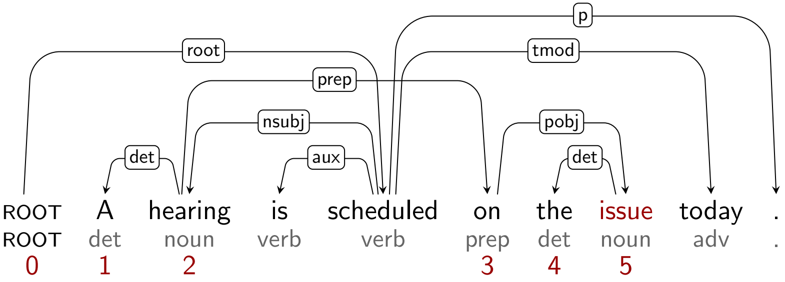
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



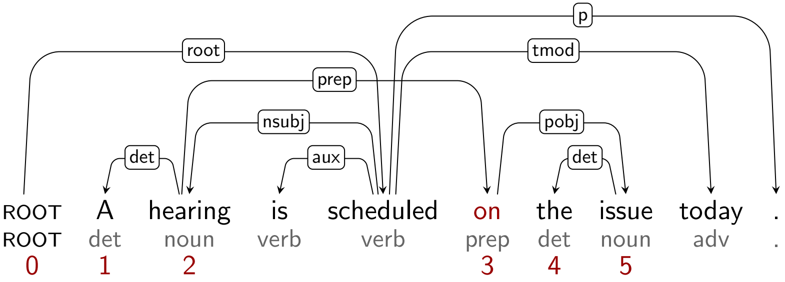
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



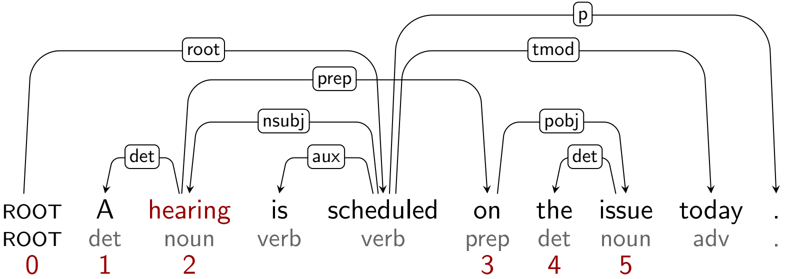
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



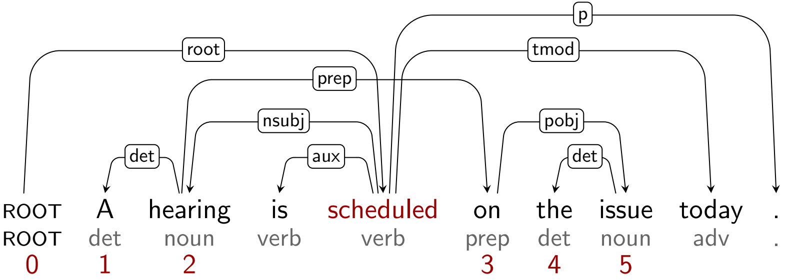
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



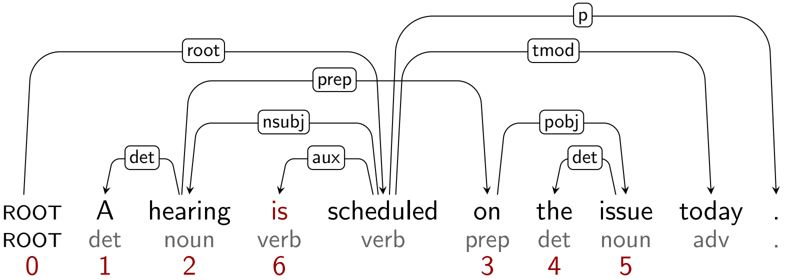
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



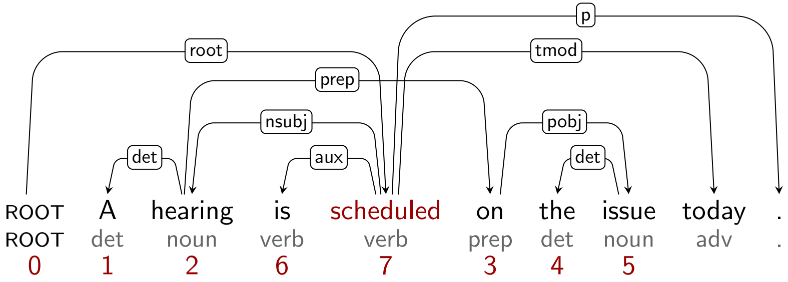
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



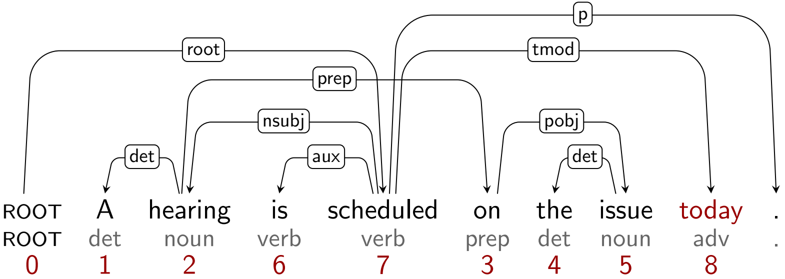
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



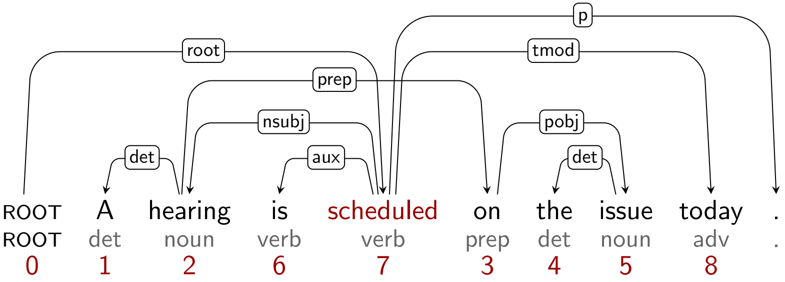
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



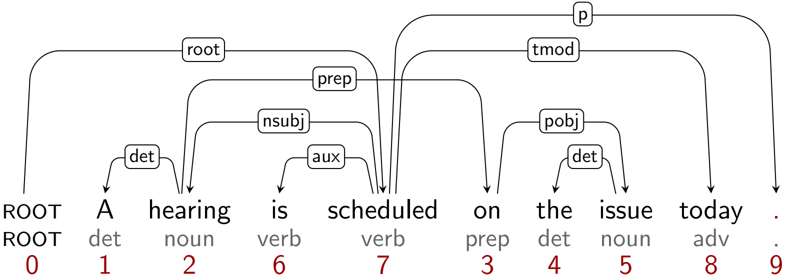
Projectivity and Word Order

- 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
  - 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 $<$ [Veselá et al. 2004]



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
  - 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 $<$ [Veselá et al. 2004]



Transition System for Online Reordering

Configuration: (S, B, A) $[S = \text{Stack},\ B = \text{Buffer},\ A = \text{Arcs}]$

Initial: $([\ ], [0, 1, \ldots, n], \{\ \})$

Terminal: $([0], [\ ], A)$

Shift: $(S, i|B, A) \quad \Rightarrow \quad (S|i, B, A)$

Right-Arc(k): $(S|i|j, B, A) \quad \Rightarrow \quad (S|i, B, A \cup \{(i, j, k)\})$

Left-Arc(k): $(S|i|j, B, A) \quad \Rightarrow \quad (S|j, B, A \cup \{(j, i, k)\}) \qquad i \neq 0$

Swap: $(S|i|j, B, A) \quad \Rightarrow \quad (S|j, i|B, A) \qquad 0 < i < j$

Transition System for Online Reordering

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

- Transition-based parsing with two interleaved processes:
  1. Sort words into projective order $<_p$
  2. Build tree T by connecting adjacent subtrees
- T is projective with respect to $<_p$ but not (necessarily) $<$

Example Transition Sequence

$[\ ]_S$ [ROOT, A, hearing, is, scheduled, on, the, issue, today, .$]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

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

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

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

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

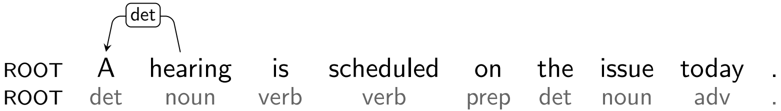
Example Transition Sequence

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

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

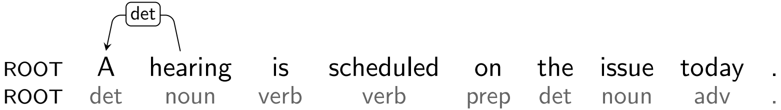
Example Transition Sequence

$[\text{ROOT, hearing}]_S$ $[\text{is, scheduled, on, the, issue, today, .}]_B$



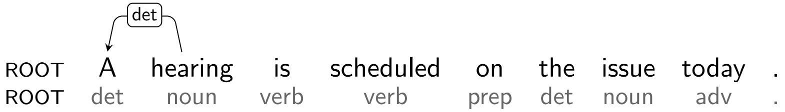
Example Transition Sequence

[ROOT, hearing, is]$_S$ [scheduled, on, the, issue, today, .]$_B$



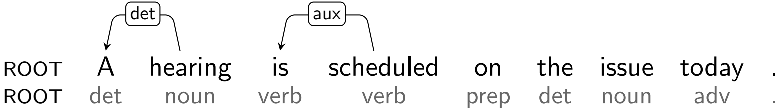
Example Transition Sequence

[ROOT, hearing, is, scheduled]$_S$ [on, the, issue, today, .]$_B$



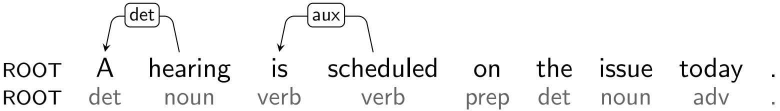
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [on, the, issue, today, .]$_B$



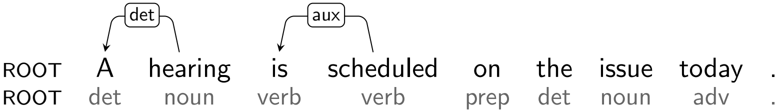
Example Transition Sequence

$[\text{ROOT, hearing, scheduled, on}]_S \quad [\text{the, issue, today, .}]_B$



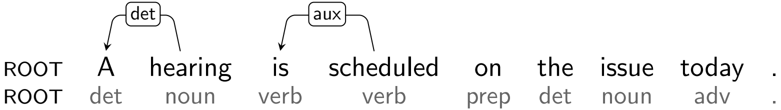
Example Transition Sequence

$[$ROOT, hearing, scheduled, on, the$]_S$ $[$issue, today, .$]_B$



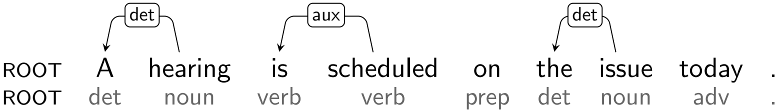
Example Transition Sequence

$[\text{ROOT, hearing, scheduled, on, the, issue}]_S$ $[\text{today, .}]_B$



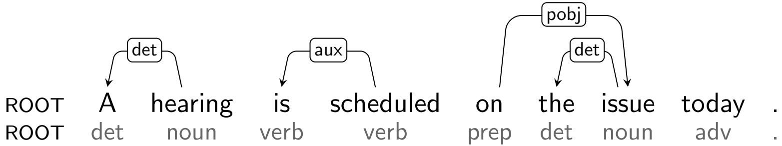
Example Transition Sequence

[ROOT, hearing, scheduled, on, issue]$_S$ [today, .]$_B$



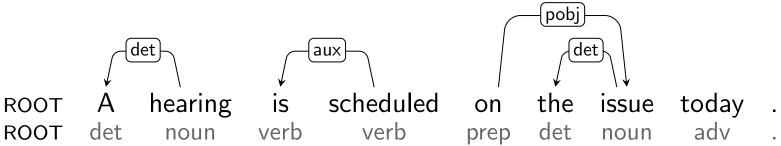
Example Transition Sequence

$[\text{ROOT, hearing, scheduled, on}]_S$ $[\text{today, .}]_B$



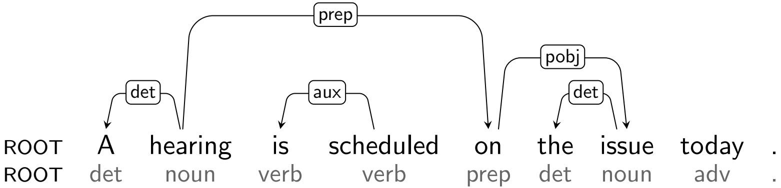
Example Transition Sequence

[ROOT, hearing, on]$_S$ [scheduled, today, .]$_B$



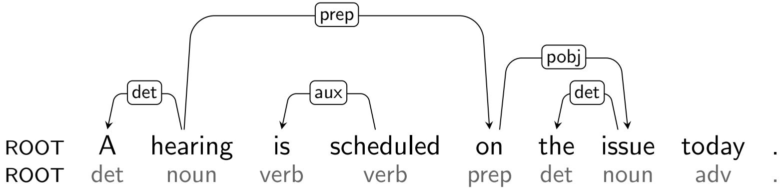
Example Transition Sequence

$[\text{ROOT, hearing}]_S$ $[\text{scheduled, today, .}]_B$



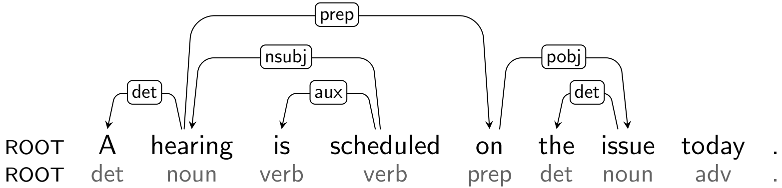
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [today, .]$_B$



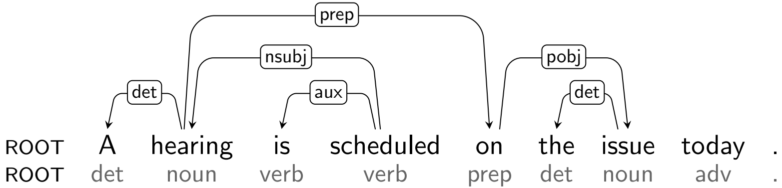
Example Transition Sequence

[ROOT, scheduled]$_S$ [today, .]$_B$



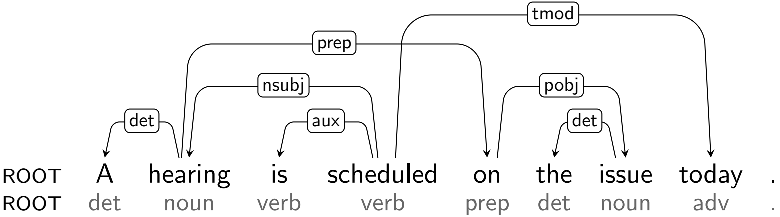
Example Transition Sequence

[ROOT, scheduled, today]$_S$ [.]$_B$



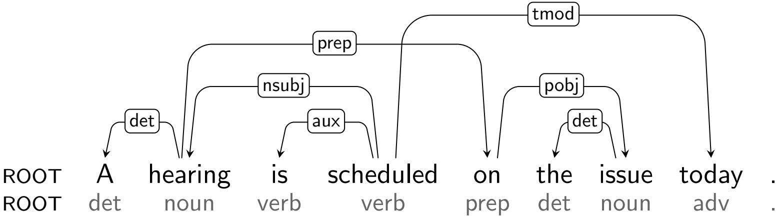
Example Transition Sequence

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



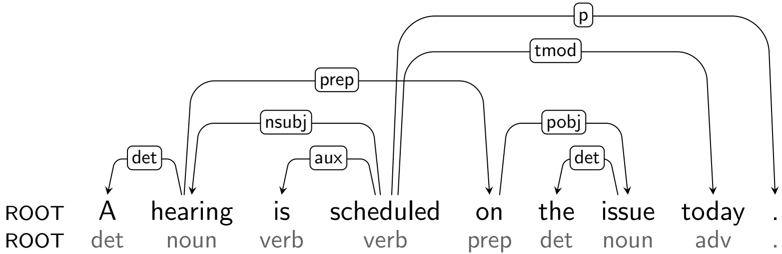
Example Transition Sequence

$[\text{ROOT, scheduled, .}]_S \quad [\ ]_B$



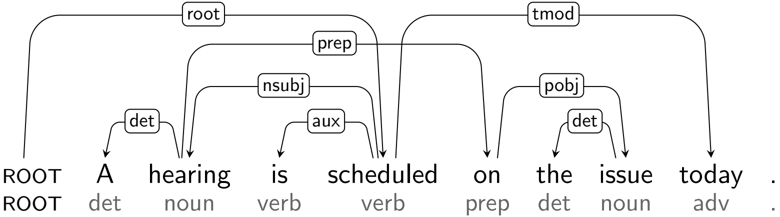
Example Transition Sequence

[ROOT, scheduled]$_S$ [ ]$_B$



Example Transition Sequence

$[\text{ROOT}]_S \quad [\ ]_B$



Analysis

- Correctness:
  - Sound and complete for the class of non-projective trees
- Complexity for greedy or beam search parsing:
  - Quadratic running time in the worst case
  - Linear running time in the average case
- Works well with beam search and structured prediction

	Czech		German	
	LAS	UAS	LAS	UAS
Projective	80.8	86.3	86.2	88.5
Reordering	83.9	89.1	88.7	90.9

[Bohnet and Nivre 2012]

Morphology and Syntax

- Morphological analysis in dependency parsing:
  - Crucially assumed as input, not predicted by the parser
  - Pipeline approach may lead to error propagation
  - Most PCFG-based parsers at least predict their own tags
- Recent interest in joint models for morphology and syntax:
  - Graph-based [McDonald 2006, Lee et al. 2011, Li et al. 2011]
  - Transition-based [Hatori et al. 2011, Bohnet and Nivre 2012]
- Can improve both morphology and syntax

Transition System for Morphology and Syntax

Configuration: (S, B, M, A) [M = Morphology]

Initial: $([\ ], [0, 1, \ldots, n], \{\ \}, \{\ \})$

Terminal: $([0], [\ ], M, A)$

Shift(p): $(S, i|B, M, A) \Rightarrow (S|i, B, M \cup \{(i, m)\}, A)$

Right-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|i, B, M, A \cup \{(i, j, k)\})$

Left-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|j, B, M, A \cup \{(j, i, k)\})$ $i \neq 0$

Swap: $(S|i|j, B, M, A) \Rightarrow (S|j, i|B, M, A)$ $0 < i < j$

Transition System for Morphology and Syntax

Configuration: (S, B, M, A) $[M = \text{Morphology}]$

Initial: $([\,], [0, 1, \ldots, n], \{\,\}, \{\,\})$

Terminal: $([0], [\,], M, A)$

Shift(p): $(S, i|B, M, A) \Rightarrow (S|i, B, M \cup \{(i, m)\}, A)$

Right-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|i, B, M, A \cup \{(i, j, k)\})$

Left-Arc(k): $(S|i|j, B, M, A) \Rightarrow (S|j, B, M, A \cup \{(j, i, k)\})$ $i \neq 0$

Swap: $(S|i|j, B, M, A) \Rightarrow (S|j, i|B, M, A)$ $0 < i < j$

- Transition-based parsing with three interleaved processes:
  - Assign morphology when words are shifted onto the stack
  - Optionally sort words into projective order $<_p$
  - Build dependency tree T by connecting adjacent subtrees

Parsing Richly Inflected Languages

- Full morphological analysis: lemma + postag + features
  - Beam search and structured predication
  - Parser selects from k best tags + features
  - Rule-based morphology provides additional features
- Evaluation metrics:
  - PM = morphology (postag + features)
  - LAS = labeled attachment score

	Czech		Finnish		German		Hungarian		Russian	
	PM	LAS	PM	LAS	PM	LAS	PM	LAS	PM	LAS
Pipeline	93.0	83.1	88.8	79.9	89.1	91.8	96.1	88.4	92.6	87.4
Joint	94.4	83.5	91.6	82.5	91.2	92.1	97.4	89.1	95.1	88.0

[Bohnet et al. 2013]

Coming Up Next

1. Basic notions of dependency grammar and dependency parsing
2. Graph-based and transition-based dependency parsing
3. Advanced graph-based parsing techniques
4. Advanced transition-based parsing techniques
5. Neural network techniques in dependency parsing
6. Multilingual parsing from raw text to universal dependencies

References and Further Reading

- 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.
- Bernd Bohnet and Jonas Kuhn. 2012.
  The best of both worlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chpater of the Association for Computational Linguistics (EACL)*, pages 77–87.
- Bernd Bohnet and Joakim Nivre. 2012.
  A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465.
- Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013.
  Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.
- Jinho D. Choi and Andrew McCallum. 2013.

Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062.

- Jinho D. Choi and Martha Palmer. 2011.
  Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 687–692.

- Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011.
  Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1245.

- Michael Collins and Brian Roark. 2004.
  Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 112–119.

- Michael Collins. 2002.
  Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

- Michael A. Covington. 2001.
  A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.
- Yoav Goldberg and Michael Elhadad. 2010.
  An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 742–750.
- Yoav Goldberg and Joakim Nivre. 2012.
  A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976.
- Yoav Goldberg and Joakim Nivre. 2013.
  Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.
- Carlos Gómez-Rodríguez and Joakim Nivre. 2010.
  A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501.
- Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011.

Incremental joint pos tagging and dependency parsing in chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing (IJCNLP)*, pages 1216–1224.

- Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013.
  A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172.

- Liang Huang and Kenji Sagae. 2010.
  Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1077–1086.

- Liang Huang, Suphan Fayong, and Yang Guo. 2012.
  Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151.

- Richard Johansson and Pierre Nugues. 2006.
  Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 206–210.

- Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011.
  Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 673–682.

- John Lee, Jason Naradowsky, and David A. Smith. 2011.
  A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 885–894.

- Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011.
  Joint models for chinese pos tagging and dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1180–1191.

- Ryan McDonald. 2006.
  Discriminative Training and Spanning Tree Algorithms for Dependency Parsing. University of Pennsylvania. Ph.D. thesis, PhD Thesis.

- 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, Marco Kuhlmann, and Johan Hall. 2009.
  An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76.
- 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.
- Joakim Nivre. 2009.
  Non-projective dependency parsing in expected linear time. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 351–359.
- Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013.
  A transition-based dependency parser using a dynamic parsing strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 135–144.
- Katerina Veselá, Havelka Jiri, and Eva Hajicová. 2004.

Condition of projectivity in the underlying dependency structures. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 289–295.

- Anssi Yli-Jyrä. 2003.
  Multiplanarity – a model for dependency structures in treebanks. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT)*, pages 189–200.
- Yue Zhang and Stephen Clark. 2008.
  A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571.
- Yue Zhang and Joakim Nivre. 2011.
  Transition-based parsing with rich non-local features. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 188–193.
- Yue Zhang and Joakim Nivre. 2012.
  Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 1391–1400.