Introduction to Data-Driven Dependency Parsing

Introductory Course, ESSLLI 2007

Ryan McDonald[1] Joakim Nivre[2]

[1]Google Inc., New York, USA
E-mail: ryanmcd@google.com

[2]Uppsala University and Växjö University, Sweden
E-mail: nivre@msi.vxu.se

Overview of the Course

- Dependency parsing (Joakim)
- Machine learning methods (Ryan)
- **Transition-based models** (Joakim)
- Graph-based models (Ryan)
- Loose ends (Joakim, Ryan):
  - Other approaches
  - Empirical results
  - Available software

Notation Reminder

- Sentence $x = w_0, w_1, \ldots, w_n$, with $w_0 = root$
- $L = \{l_1, \ldots, l_{|L|}\}$ set of permissible arc labels
- Let $G = (V, A)$ be a dependency graph for sentence $\boldsymbol{x}$ where:
  - $V = \{0, 1, \ldots, n\}$ is the vertex set
  - A is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from w_i to w_j with label $l_k \in L$
- By the usual definition, G **is a tree**

Data-Driven Parsing

- Goal: Learn a good predictor of dependency graphs
- Input: x
- Output: dependency graph/tree G
- This lecture:
  - Parameterize parsing by transitions
  - Learn to predict transitions given the input and a history
  - Predict new graphs using deterministic parsing algorithm
- Next lecture:
  - Parameterize parsing by dependency arcs
  - Learn to predict entire graphs given the input
  - Predict new graphs using spanning tree algorithms

Lecture 3: Outline

- Transition systems
- Deterministic classifier-based models
  - Parsing algorithm
  - Stack-based and list-based transition systems
  - Classifier-based parsing
- Pseudo-projective parsing

Transition Systems

- A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where
  1. C is a set of configurations, each of which contains a buffer β of (remaining) nodes and a set A of dependency arcs,
  2. T is a set of transitions, each of which is a (partial) function $t : C \rightarrow C$,
  3. c_s is an initialization function, mapping a sentence $x = w_0, w_1, \ldots, w_n$ to a configuration with $\beta = [1, \ldots, n]$,
  4. $C_t \subseteq C$ is a set of terminal configurations.
- Note:
  - A configuration represents a parser state.
  - A transition represents a parsing action (parser state update).

Transition Sequences

- Let $S = (C, T, c_s, C_t)$ be a transition system.
- A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in S is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that
  1. $c_0 = c_s(x)$,
  2. $c_m \in C_t$,
  3. for every i $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.
- The parse assigned to x by $C_{0,m}$ is the dependency graph $G_{c_m} = (\{0, 1, \ldots, n\}, A_{c_m})$, where A_{c_m} is the set of dependency arcs in c_m.

Deterministic Parsing

- An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \rightarrow T$.
- Given a transition system $S = (C, T, c_s, C_t)$ and an oracle o, deterministic parsing can be achieved by the following simple algorithm:

$$
\begin{array}{l}
\text{Parse}(x = (w_0, w_1, \ldots, w_n)) \\
\quad c \leftarrow c_s(x) \\
\quad \textbf{while } c \notin C_t \\
\qquad\quad c = [o(c)](c) \\
\quad \textbf{return } G_c
\end{array}
$$

- NB: Oracles can be approximated by classifiers (cf. lecture 2).

Stack-Based Transition Systems

- A stack-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a triple $c = (\sigma, \beta, A)$, where
  1. σ is a stack of tokens $i \leq m$ (for some $m \leq n$),
  2. β is a buffer of tokens $j > m$,
  3. A is a set of dependency arcs such that $G = (\{0, 1, \ldots, n\}, A)$ is a dependency graph for x.
- A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where
  1. C is the set of all stack-based configurations,
  2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset)$,
  3. T is a set of transitions, each of which is a function $t : C \rightarrow C$,
  4. $C_t = \{c \in C | c = (\sigma, [\,], A)\}$.
- Notation:
  - $\sigma|i$ = stack with top i ($|$ left-associative)
  - $i|\beta$ = buffer with next token i ($|$ right-associative)

Shift-Reduce Dependency Parsing

- Transitions:
  - Left-Arc$_k$:

    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$
  - Right-Arc$_k$:

    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j, k)\})$$
  - Shift:

    $$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$
- Preconditions:
  - Left-Arc$_k$:

    $$\neg[i = 0]$$
    $$\neg\exists i' \exists k' [(i', i, k') \in A]$$
  - Right-Arc$_k$:

    $$\neg\exists i' \exists k' [(i', j, k') \in A]$$

Example: Shift-Reduce Parsing

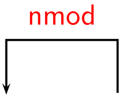
$[\text{root}_0]_\sigma$ $[\text{Economic}_1$ news_2 had_3 little_4 effect_5 on_6 financial_7 markets_8 $._9]_\beta$

Example: Shift-Reduce Parsing

$[\text{root}_0\ \text{Economic}_1]_\sigma\ [\text{news}_2\ \text{had}_3\ \text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ ._9]_\beta$

Shift

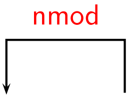
Example: Shift-Reduce Parsing



[root$_0$]$_\sigma$ Economic$_1$ [news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$.$_9$]$_\beta$

Left-Arc$_{nmod}$

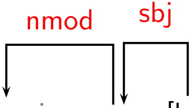
Example: Shift-Reduce Parsing



$[$root$_0$ Economic$_1$ news$_2]_\sigma$ $[$had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$.$_9]_\beta$

Shift

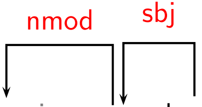
Example: Shift-Reduce Parsing



$[\text{root}_0]_\sigma$ Economic_1 news_2 $[\text{had}_3\ \text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ ._9]_\beta$

Left-Arc$_{sbj}$

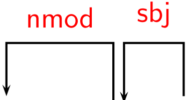
Example: Shift-Reduce Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$]$_\sigma$ [little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$.$_9$]$_\beta$

Shift

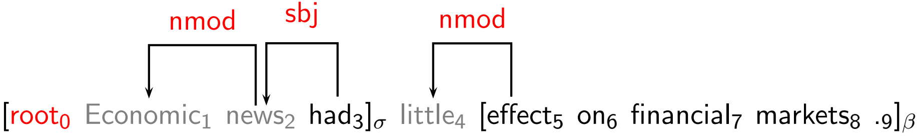
Example: Shift-Reduce Parsing



$[\text{root}_0\ \text{Economic}_1\ \text{news}_2\ \text{had}_3\ \text{little}_4]_\sigma\ [\text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ ._9]_\beta$

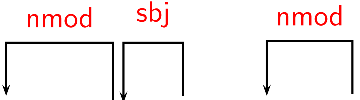
Shift

Example: Shift-Reduce Parsing



Left-Arc$_{nmod}$

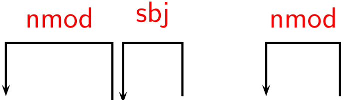
Example: Shift-Reduce Parsing



[root_0 Economic_1 news_2 had_3 little_4 effect_5]$_\sigma$ [on_6 financial_7 markets_8 $._9$]$_\beta$

Shift

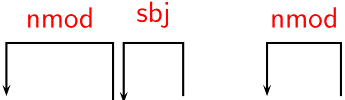
Example: Shift-Reduce Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$]$_\sigma$ [financial$_7$ markets$_8$.$_9$]$_\beta$

Shift

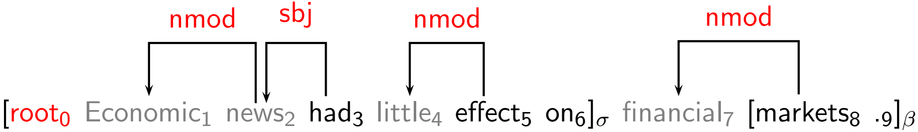
Example: Shift-Reduce Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$]$_\sigma$ [markets$_8$.$_9$]$_\beta$

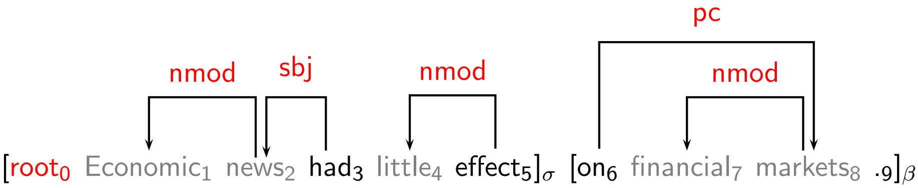
Shift

Example: Shift-Reduce Parsing



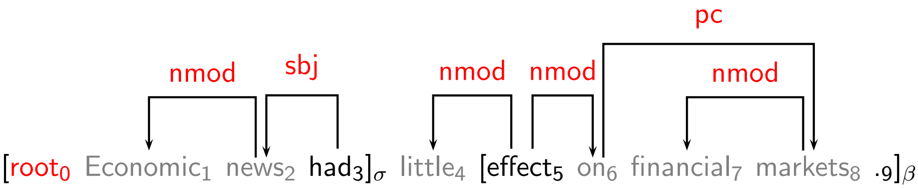
Left-Arc$_{nmod}$

Example: Shift-Reduce Parsing



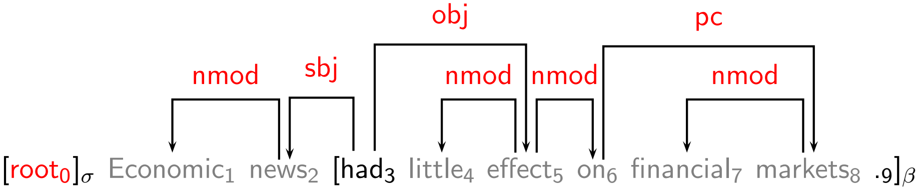
Right-Arc$_{pc}$

Example: Shift-Reduce Parsing



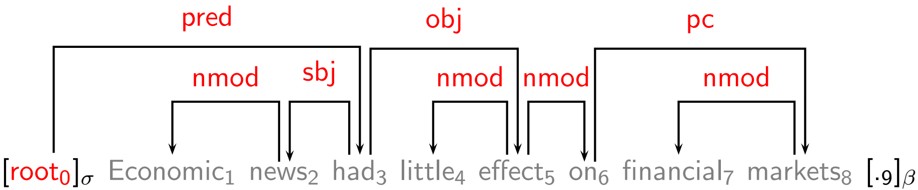
Right-Arc$_{nmod}$

Example: Shift-Reduce Parsing



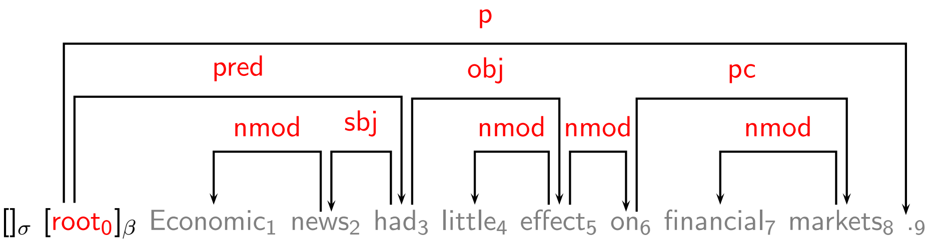
Right-Arc$_{obj}$

Example: Shift-Reduce Parsing



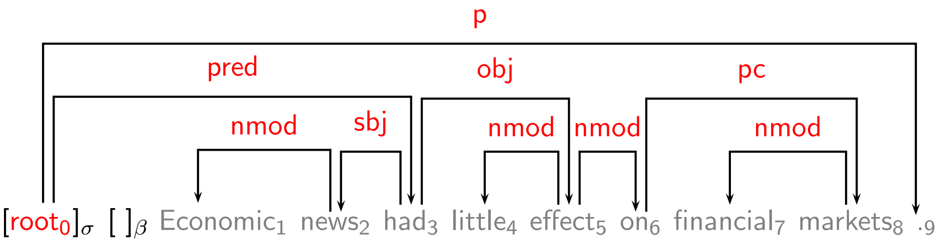
Right-Arc$_{pred}$

Example: Shift-Reduce Parsing



Right-Arc$_p$

Example: Shift-Reduce Parsing



Shift

Theoretical Results

- Complexity:
  - Deterministic shift-reduce parsing has time and space complexity $O(n)$, where n is the length of the input sentence.
- Correctness:
  - For every transition sequence $C_{0,m}$, G_{c_m} is a projective dependency forest (soundness).
  - For every projective dependency forest G, there is a transition sequence $C_{0,m}$ such that $G_{c_m} = G$ (completeness).
- Note:
  - A dependency forest is (here) a dependency graph satisfying Root, Single-Head, and Acyclicity (but not Connectedness).
  - A dependency forest $G = (V, A)$ can be transformed into a dependency tree by adding arcs of the form $(0, i, k)$ (for some $l_k \in L$) for every root $i \in V$ ($i \neq 0$).

Variations on Shift-Reduce Parsing

- Empty stack initialization:
  - If we can assume that there is only one node i such that $(0, i, k) \in A$, then we can reduce ambiguity by starting with an empty stack (and adding the arc $(0, i, k)$ after termination).
- Iterative parsing [Yamada and Matsumoto 2003]:
  - Same transition system (with empty stack initialization)[1]
  - Given a terminal configuration:
    - $(\sigma, [\,], A) \Longrightarrow ([\,], \sigma, A)$
    - Terminate when A has not been modified in the last iteration.
- Modified transition systems:
  - Arc-eager parsing [Nivre 2003]
  - Non-projective parsing [Attardi 2006]

[1] NB: Left-Arc $\Rightarrow$ Right, Right-Arc $\Rightarrow$ Left

Arc-Eager Parsing

- Transitions:
  - Left-Arc$_k$:
    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$
  - Right-Arc$_k$:
    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, j, k)\})$$
  - Reduce:
    $$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$$
  - Shift:
    $$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$
- Preconditions:
  - Left-Arc$_k$:
    $$\neg[i = 0]$$
    $$\neg\exists i' \exists k' [(i', i, k') \in A]$$
  - Right-Arc$_k$:
    $$\neg\exists i' \exists k' [(i', j, k') \in A]$$
  - Reduce:
    $$\exists i' \exists k' [(i', i, k') \in A]$$

Example: Arc-Eager Parsing

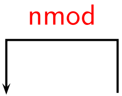
$[\text{root}_0]_\sigma$ $[\text{Economic}_1$ news_2 had_3 little_4 effect_5 on_6 financial_7 markets_8 $._9]_\beta$

Example: Arc-Eager Parsing

$[\text{root}_0\ \text{Economic}_1]_\sigma\ [\text{news}_2\ \text{had}_3\ \text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ ._9]_\beta$

Shift

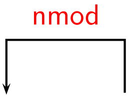
Example: Arc-Eager Parsing



[root$_0$]$_\sigma$ Economic$_1$ [news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$.$_9$]$_\beta$

Left-Arc$_{nmod}$

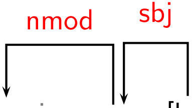
Example: Arc-Eager Parsing



[root$_0$ Economic$_1$ news$_2$]$_\sigma$ [had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$.$_9$]$_\beta$

Shift

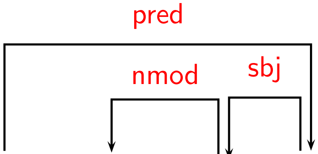
Example: Arc-Eager Parsing



$[\text{root}_0]_\sigma$ Economic_1 news_2 $[\text{had}_3$ little_4 effect_5 on_6 financial_7 markets_8 $._9]_\beta$

Left-Arc_{sbj}

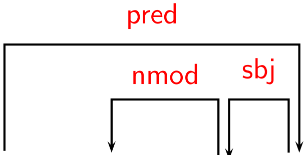
Example: Arc-Eager Parsing



$[\text{root}_0\ \text{Economic}_1\ \text{news}_2\ \text{had}_3]_\sigma\ [\text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ ._9]_\beta$

Right-Arc$_{pred}$

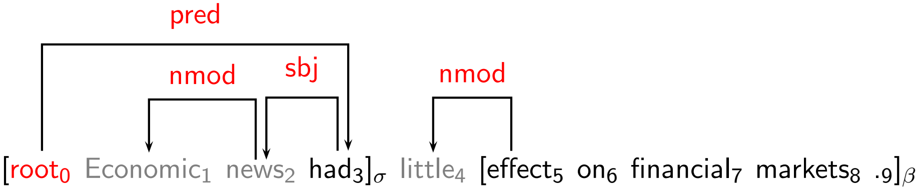
Example: Arc-Eager Parsing



$[\text{root}_0$ Economic_1 news_2 had_3 $\text{little}_4]_\sigma$ $[\text{effect}_5$ on_6 financial_7 markets_8 $._9]_\beta$

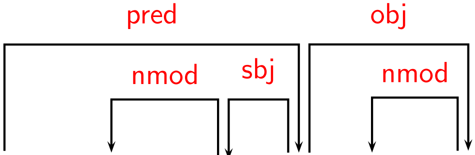
Shift

Example: Arc-Eager Parsing



Left-Arc$_{nmod}$

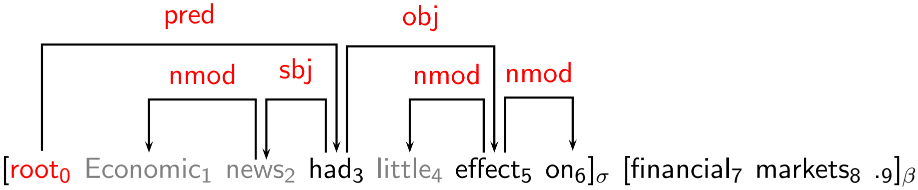
Example: Arc-Eager Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$]$_\sigma$ [on$_6$ financial$_7$ markets$_8$.$_9$]$_\beta$

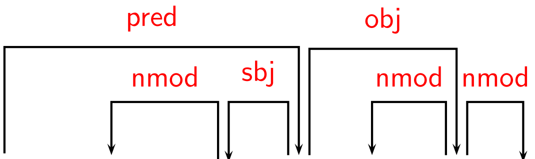
Right-Arc$_{obj}$

Example: Arc-Eager Parsing



Right-Arc$_{nmod}$

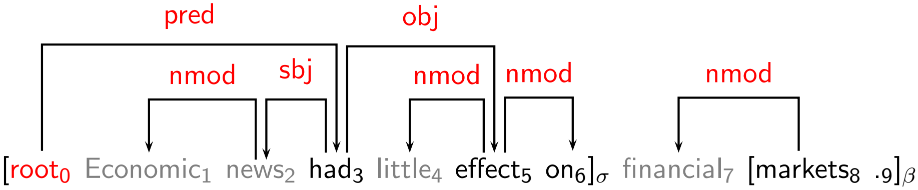
Example: Arc-Eager Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$]$_\sigma$ [markets$_8$.$_9$]$_\beta$

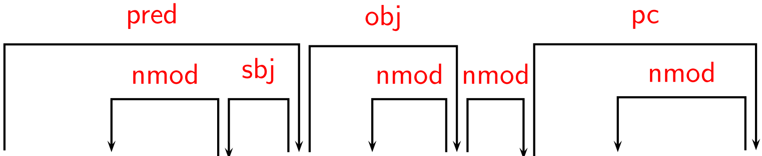
Shift

Example: Arc-Eager Parsing



Left-Arc$_{nmod}$

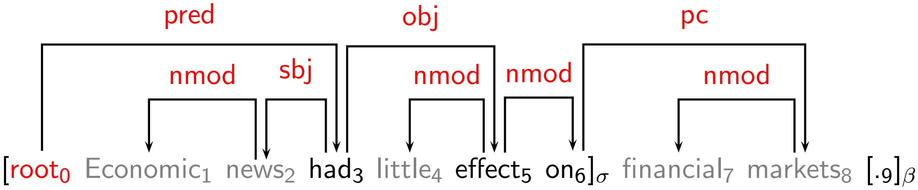
Example: Arc-Eager Parsing



[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$ on$_6$ financial$_7$ markets$_8$]$_\sigma$ [.$_9$]$_\beta$

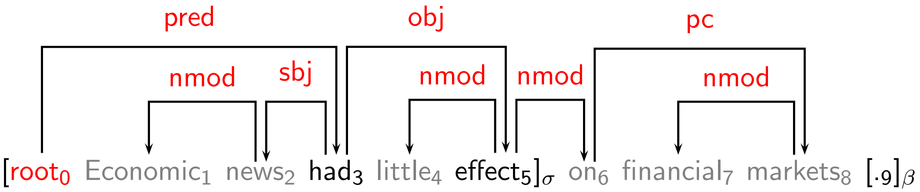
Right-Arc$_{pc}$

Example: Arc-Eager Parsing



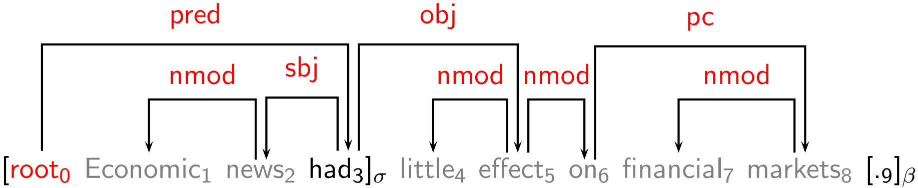
Reduce

Example: Arc-Eager Parsing



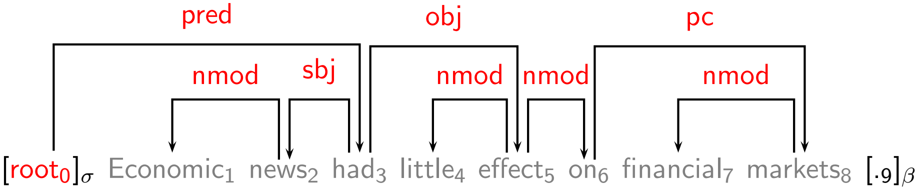
Reduce

Example: Arc-Eager Parsing



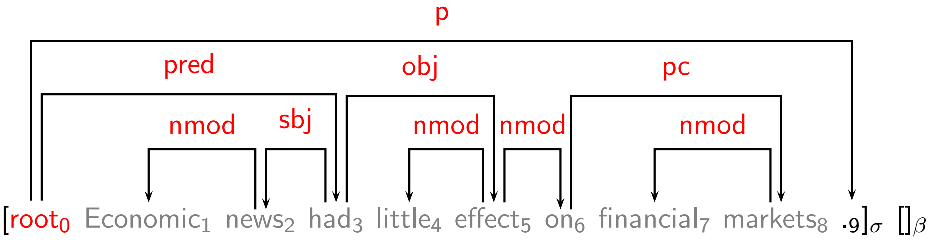
Reduce

Example: Arc-Eager Parsing



Reduce

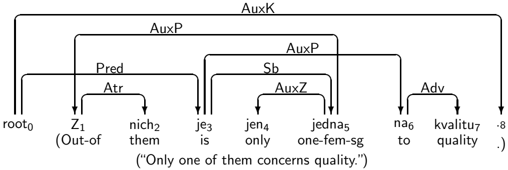
Example: Arc-Eager Parsing



Right-Arc$_p$

Non-Projective Parsing

- New transitions:
  - NP-Left-Arc$_k$:
    $$(\sigma|i|i', j|\beta, A) \Rightarrow (\sigma|i', j|\beta, A \cup \{(j, i, k)\})$$
  - NP-Right-Arc$_k$:
    $$(\sigma|i|i', j|\beta, A) \Rightarrow (\sigma|i, i'|\beta, A \cup \{(i, j, k)\})$$
- Handles most naturally occurring non-projective dependency relations (94% in the Prague Dependency Treebank).



- More expressive extensions are possible [Attardi 2006].

Comparing Algorithms

- Expressivity:
  - Arc-standard and arc-eager shift-reduce parsing is limited to projective depedendency graphs.
  - Simple extensions can handle a subset of non-projective dependency graphs.
- Complexity:
  - Space complexity is $O(n)$ for all deterministic parsers (even with simple extensions).
  - Time complexity is $O(n)$ for single-pass parsers, $O(n^2)$ for iterative parsers.
- More complex extensions to handle non-projective dependency graphs will affect time complexity.

List-Based Transition Systems

- A list-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a quadruple $c = (\lambda_1, \lambda_2, \beta, A)$, where
  1. λ_1 is a list of tokens $i_1 \leq m_1$ (for some $m_1 \leq n$),
  2. λ_2 is a list of tokens $i_2 \leq m_2$ (for some m_2, $m_1 < m_2 \leq n$),
  3. β is a buffer of tokens $j > m_2$,
  4. A is a set of dependency arcs such that $G = (\{0, 1, \ldots, n), A)$ is a dependency graph for x.
- A list-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where
  1. C is the set of all list-based configurations,
  2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [\,], [1, \ldots, n], \emptyset)$,
  3. T is a set of transitions, each of which is a function $t : C \rightarrow C$,
  4. $C_t = \{c \in C | c = (\lambda_1, \lambda_2, [\,], A)\}$.
- Notation:
  - $\lambda_1 | i$ = list with head i and tail λ_1 ($|$ left-associative)
  - $i | \lambda_2$ = i and tail λ_2 ($|$ right-associative)

Non-Projective Parsing

- Transitions:
  - Left-Arc$_k$:
    $$(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup \{(j, i, k)\})$$
  - Right-Arc$_k$:
    $$(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup \{(i, j, k)\})$$
  - No-Arc:
    $$(\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)$$
  - Shift:
    $$(\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1.\lambda_2|i, [\,], \beta, A)$$
- Preconditions:
  - Left-Arc:
    $$\neg[i = 0]$$
    $$\neg\exists i'\exists k'[(i', k', i) \in A]$$
    $$\neg[i \rightarrow^* j]_A$$
  - Right-Arc:
    $$\neg\exists i'\exists k'[(i', k', j) \in A]$$
    $$\neg[j \rightarrow^* i]_A$$

Projective Parsing

- Transitions:
  - Left-Arc$_k$:
    $$(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, \lambda_2, j|\beta, A \cup \{(j, i, k)\})$$
  - Right-Arc$_k$:
    $$(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1|i|j, [\,], \beta, A \cup \{(i, k, j)\})$$
  - No-Arc:
    $$(\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)$$
  - Shift:
    $$(\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1.\lambda_2|i, [\,], \beta, A)$$
- Preconditions:
  - Left-Arc:
    $$\neg[i = 0]$$
    $$\neg\exists i' \exists k' [(i', k', i) \in A]$$
  - Right-Arc:
    $$\neg\exists i' \exists k' [(i', k', j) \in A]$$
  - No-Arc:
    $$\exists i' \exists k [(i', k, i) \in A]$$

Theoretical Results

- Complexity:
  - Deterministic list-based parsing has time complexity $O(n^2)$ and space complexity $O(n)$, where n is the length of the input sentence.
- Correctness:
  - For every transition sequence $C_{0,m}$, G_{c_m} is a (projective) dependency forest (soundness).
  - For every (projective) dependency forest G, there is a transition sequence $C_{0,m}$ such that $G_{c_m} = G$ (completeness).

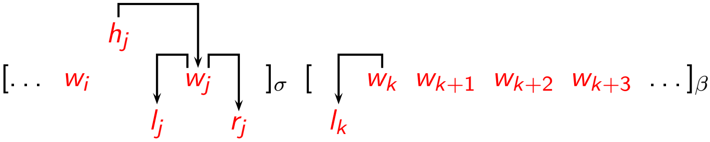
Classifier-Based Parsing

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.
- Learning problem:
  - Approximate a function from configurations (represented by feature vectors) to transitions, given a training set of (gold standard) transition sequences.
  - Three issues:
    - How do we represent configurations by feature vectors?
    - How do we derive training data from treebanks?
    - How do we learn classifiers?

Feature Representations

- A feature representation $\mathbf{f}(c)$ of a configuration c is a vector of simple features $\mathbf{f}_i(c)$.
- Typical features are defined in terms of attributes of nodes in the dependency graph.
  - Nodes:
    - Target nodes (top of σ, head of λ_1, λ_2, β)
    - Linear context (neighbors in σ, λ_1, λ_2, or β)
    - Structural context (parents, children, siblings given A)
  - Attributes:
    - Word form (and/or lemma)
    - Part-of-speech (and morpho-syntactic features)
    - Dependency type (if labeled)
    - Distance (between target tokens)

A Typical Model [Nivre et al. 2006]



	w_i	h_j	l_j	w_j	r_j	l_k	w_k	w_{k+1}	w_{k+2}	w_{k+3}
FORM		+		+			+	+		
LEMMA				+			+			
CPOS				+			+			
POS	+			+			+	+	+	+
FEATS				+			+			
DEPREL			+	+	+	+				

Training Data

- Training instances have the form $(\mathbf{f}(c), t)$, where
  1. $\mathbf{f}(c)$ is a feature 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 x with (gold standard) dependency graph G_x, we construct a transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ such that
    1. $c_0 = c_s(x)$,
    2. $G_{c_m} = G_x$,
  - For each configuration $c_i (i < m)$, we construct a training instance $(\mathbf{f}(c_i), t_i)$, where $t_i(c_i) = c_{i+1}$.

Learning Classifiers

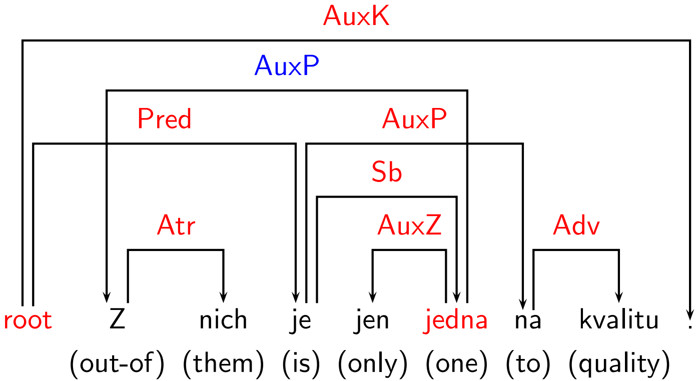
- Learning methods:
  - Support vector machines (SVM)
    [Kudo and Matsumoto 2002, Yamada and Matsumoto 2003, Isozaki et al. 2004, Cheng et al. 2004, Nivre et al. 2006]
    - Polynomial kernel ($d \geq 2$)
    - Different techniques for multiclass classification
    - Training efficiency problematic for large data sets
  - Memory-based learning (MBL)
    [Nivre et al. 2004, Nivre and Scholz 2004, Attardi 2006]
    - *k*-NN classification
    - Different distance functions
    - Parsing efficiency problematic for large data sets
  - Maximum entropy modeling (MaxEnt)
    [Cheng et al. 2005, Attardi 2006]
    - Extremely efficient parsing
    - Slightly less accurate

Pseudo-Projective Parsing

- Technique for non-projective dependency parsing with a data-driven projective parser [Nivre and Nilsson 2005].
- Four steps:
  1. Projectivize dependency graphs in training data, encoding information about transformations in augmented arc labels.
  2. Train projective parser (as usual).
  3. Parse new sentences using projective parser (as usual).
  4. Deprojectivize output dependency graphs by heuristic transformations guided by augmented arc labels.

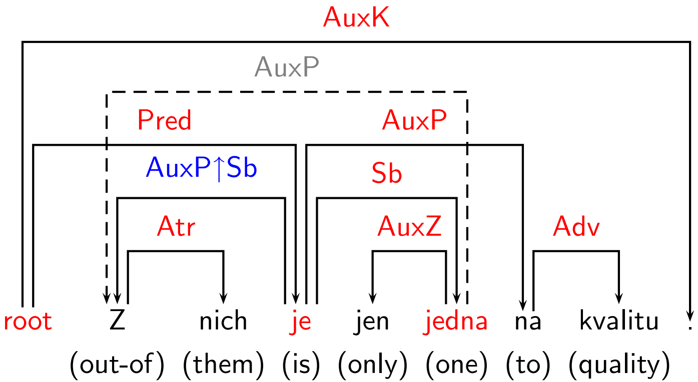
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head



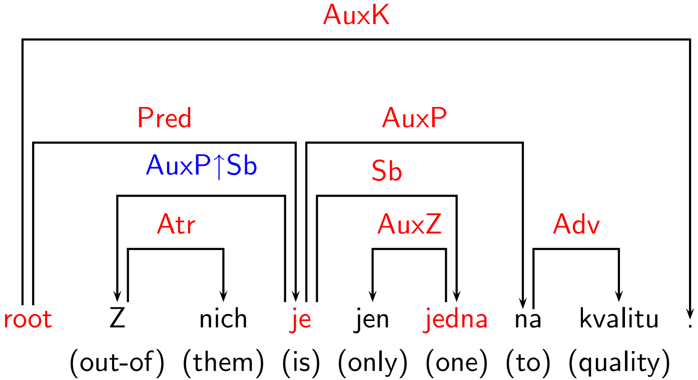
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head



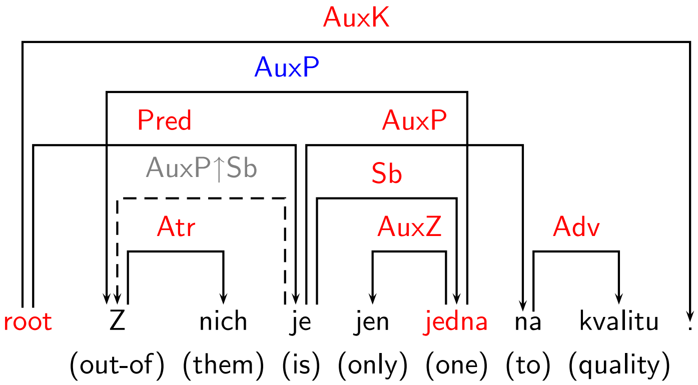
Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label



Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label



Summary – Transition-based Methods

- Transition systems
- Deterministic classifier-based parsing
  - Parsing algorithm
  - Stack-based and list-based transitions systems
  - Classifier-based parsing
- Pseudo-projective parsing

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.
- Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2004.
  Deterministic dependency structure analyzer for Chinese. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP)*, pages 500–508.
- 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.
- Hideki Isozaki, Hideto Kazawa, and Tsutomu Hirao. 2004.
  A deterministic word dependency analyzer enhanced with preference learning. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 275–281.
- 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.
- 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 and Mario Scholz. 2004.
  Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 64–70.

- Joakim Nivre, Johan Hall, and Jens Nilsson. 2004.
  Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors, *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.

- Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiugit, and Svetoslav Marinov. 2006.
  Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

- Joakim Nivre. 2003.
  An efficient algorithm for projective dependency parsing. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

- Hiroyasu Yamada and Yuji Matsumoto. 2003.

Statistical dependency analysis with support vector machines. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.