IN2110: Språkteknologiske metoder

Dependensparsing

Lilja Øvrelid

Språkteknologigruppen (LTG)

(with thanks to Stephan Oepen and Joakim Nivre)

6 april, 2022

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example

Topics for Today

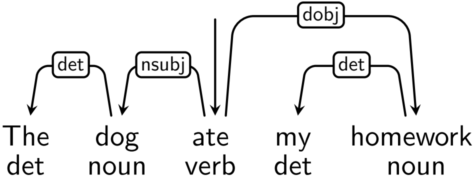
- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example
- Dependency Parser Evaluation

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example
- Dependency Parser Evaluation
- Obligatory exercise

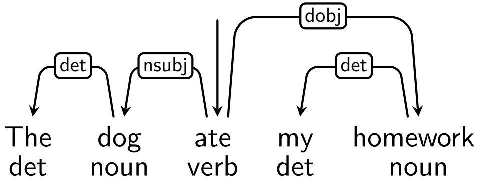
Recap: Dependency syntax

- DG is based on relationships between words, i.e., **dependency relations**
- A dependency structure can be defined as a labeled, directed graph G



Recap: Formal Conditions on Dependency Graphs

- Principles:
  - Syntactic structure is complete (Connectedness).
  - Syntactic structure is hierarchical (Acyclicity).
  - Every word has at most one syntactic head (Single-Head).



Treebanks: Universal Dependencies

universaldependencies.org

This page pertains to UD version 2.

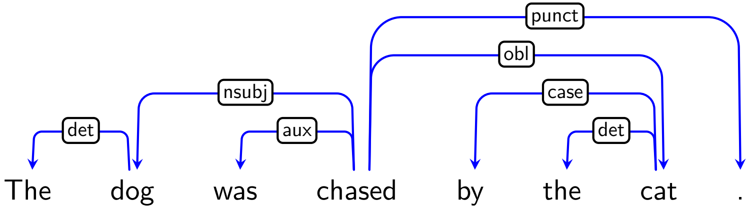
Universal Dependencies

Universal Dependencies (UD) is a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks in over 70 languages.

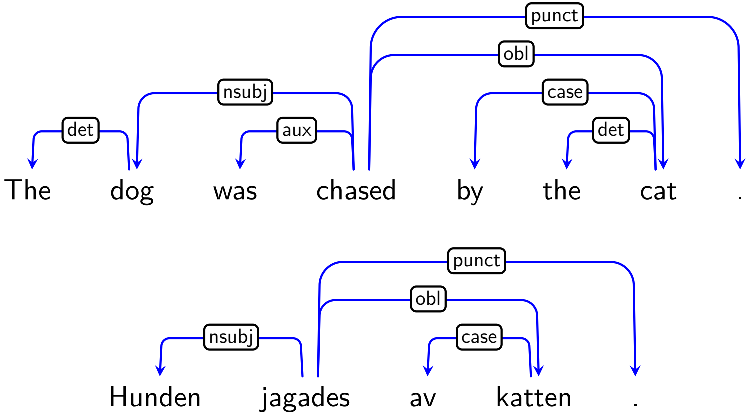
- Short introduction to UD
- UD annotation guidelines
- More information on UD:
  - How to contribute to UD
  - Tools for working with UD
  - Discussion on UD
  - UD-related events
- Query UD treebanks online:
  - SETS treebank search maintained by the University of Turku
  - PML Tree Query maintained by the Charles University in Prague
  - Kontext maintained by the Charles University in Prague
  - Grew-match maintained by Inria in Nancy
  - INESS maintained by the University of Bergen
- Download UD treebanks

If you want to receive news about Universal Dependencies, you can subscribe to the UD mailing list. If you want to discuss individual annotation

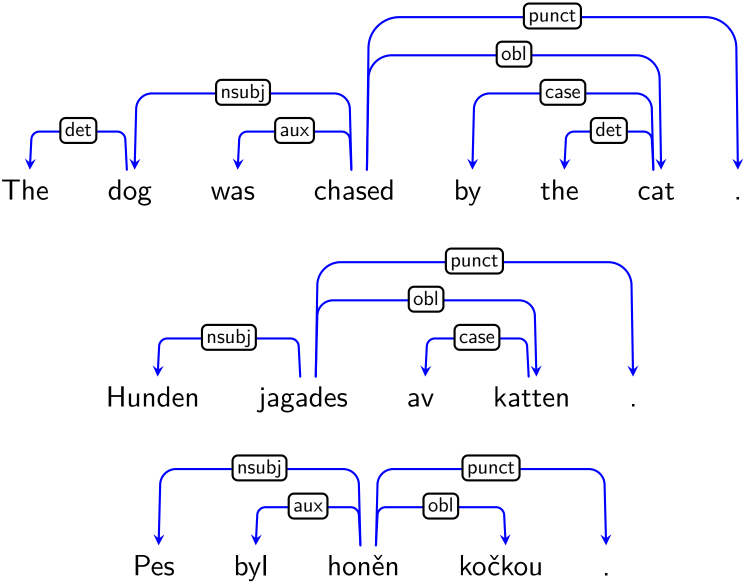
(Degrees of) Cross-Linguistic Consistency



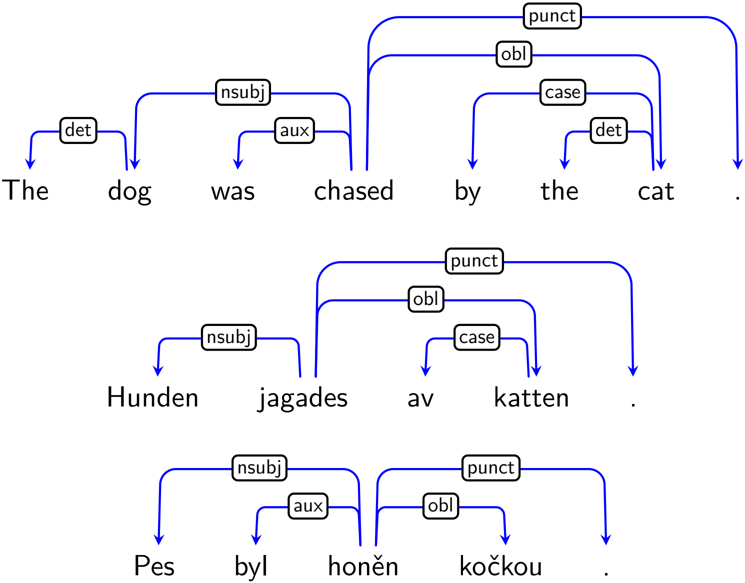
(Degrees of) Cross-Linguistic Consistency



(Degrees of) Cross-Linguistic Consistency



(Degrees of) Cross-Linguistic Consistency



- Capitalize on content words, e.g. demote case-marking prepositions.

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example
- Dependency Parser Evaluation
- Obligatory exercise

Syntactic parsing

- Automatically determining the syntactic structure for a given sentence
- Traditionally (for phrase-structure grammars):
    1. S $\rightarrow$ NP VP
    2. NP $\rightarrow$ D N
    3. VP $\rightarrow$ V NP
- search through all possible trees for a sentence
- bottom-up vs top-down approaches

Ambiguities

- more than one possible structure for a sentence
- natural languages are hugely ambiguous
- a very common problem

		PoS-ambiguities				**Attachment ambiguities**
		VB				
	VBZ	VBP	VBZ			
NNP	NNS	NN	NNS	CD	NN	
Fed	raises	interest	rates	0.5	%	in effort
						to control
						inflation

Back in the days (90s)

- Grammar-driven parsing: possible trees defined by the grammar
- Problems with **coverage**
  - only around 70% of all sentences were assigned an analysis
- Most sentences were assigned very many analyses by a grammar
  - no way of choosing between them

Enter data-driven (statistical) parsing

- Today data-driven/statistical parsing is available for a range of languages and syntactic frameworks
- Data-driven approaches: possible trees defined by the treebank
- Produce one analysis (hopefully the most likely one) for any sentence
- And get most of them correct
- Still an active field of research, improvements are still possible!

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example
- Dependency Parser Evaluation
- Obligatory exercise

Data-driven parsing

1. formal model M defining possible analyses for sentences in L
2. A sample of annotated text $S = (x_1, \ldots, x_m)$ from L
3. An inductive inference scheme I defining actual analyses for the sentences of a text $T = (x_1, \ldots, x_n)$ in L, relative to M and S.

- S is the **training data:** contains representations satisfying M
- a **treebank**: manually annotated with correct analysis
- I based on **supervised** machine learning

Data-driven dependency parsing

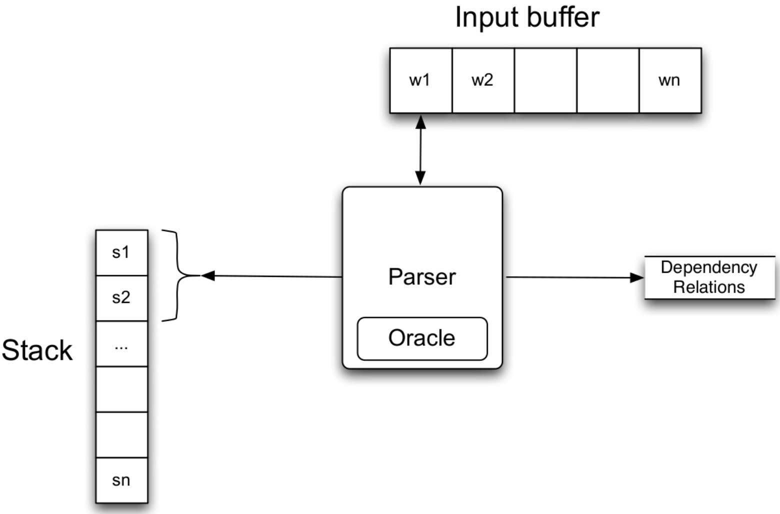
- M defined by formal conditions on dependency graphs (labeled directed graphs that are):
  - connected
  - acyclic
  - single-head
  - (projective)
- I may be defined in different ways
  - parsing method
  - machine learning algorithm, feature representations
- Two main approaches: **graph-based** and **transition-based** models
- We will focus on transition-based approaches

Transition-based approaches

Basic idea:

- define a transition system for mapping a sentence to its dependency graph
- Learning: induce a model for predicting the next state transition, given the transition history
- Parsing: Construct the optimal transition sequence, given the induced model

Architecture: Stack and Buffer Configurations



An Adaptation of Shift–Reduce Parsing

- Originally developed for non-ambiguous languages: deterministic.
- Shift ('read') tokens from input buffer, one at a time, left-to-right;
- compare top n symbols on stack, perform some action, e.g. reduce

An Adaptation of Shift–Reduce Parsing

- Originally developed for non-ambiguous languages: deterministic.
- Shift ('read') tokens from input buffer, one at a time, left-to-right;
- compare top n symbols on stack, perform some action, e.g. reduce
- Dependencies: create arcs between top of stack and front of buffer.
- Transitions:

An Adaptation of Shift–Reduce Parsing

- Originally developed for non-ambiguous languages: deterministic.
- Shift ('read') tokens from input buffer, one at a time, left-to-right;
- compare top n symbols on stack, perform some action, e.g. reduce
- Dependencies: create arcs between top of stack and front of buffer.
- Transitions:

SHIFT	move from front of buffer to top of stack
REDUCE	pop the top of stack (requires existing head)
LEFT-ARC(K)	leftward dependency of type k; reduce
RIGHT-ARC(K)	rightward dependency of type k; shift

An Adaptation of Shift–Reduce Parsing

- Originally developed for non-ambiguous languages: deterministic.
- Shift ('read') tokens from input buffer, one at a time, left-to-right;
- compare top n symbols on stack, perform some action, e.g. reduce
- Dependencies: create arcs between top of stack and front of buffer.
- Transitions:

SHIFT	move from front of buffer to top of stack
REDUCE	pop the top of stack (requires existing head)
LEFT-ARC(K)	leftward dependency of type k; reduce
RIGHT-ARC(K)	rightward dependency of type k; shift

- At REDUCE, token must be fully processed (head and dependents).
- LEFT-ARC must respect single-head constraint and unique root node.

Arc-Eager Transition System [Nivre 2003]

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

Notation: $S|i$ = stack with top i and remainder S
$j|B$ = buffer with head j and remainder B
$h(i, A)$ = i has a head in A

Example Transition Sequence

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

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

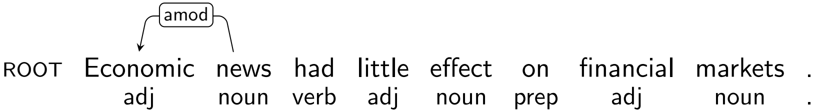
Example Transition Sequence

$[$ROOT, Economic$]_S$ $[$news, had, little, effect, on, financial, markets, .$]_B$

ROOT	Economic	news	had	little	effect	on	financial	markets	.
	adj	noun	verb	adj	noun	prep	adj	noun	.

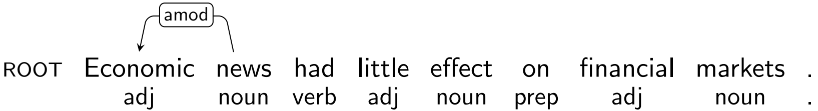
Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{news, had, little, effect, on, financial, markets, .}]_B$



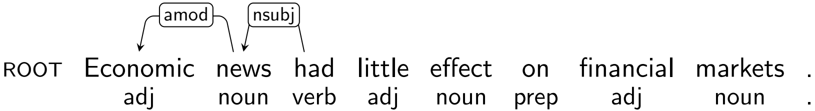
Example Transition Sequence

$[\text{ROOT, news}]_S$ $[\text{had, little, effect, on, financial, markets, .}]_B$



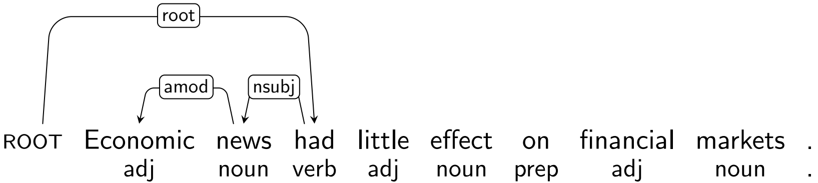
Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{had, little, effect, on, financial, markets, .}]_B$



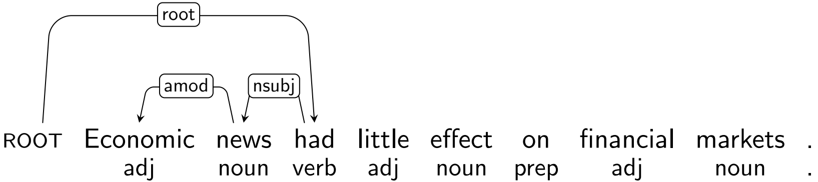
Example Transition Sequence

$[\text{ROOT, had}]_S$ $[\text{little, effect, on, financial, markets, .}]_B$



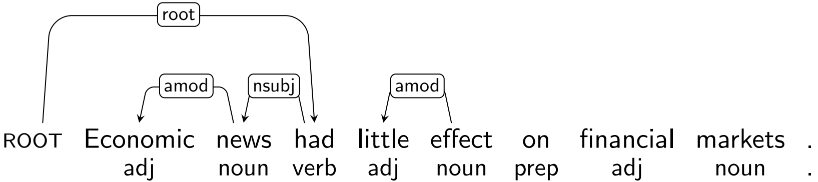
Example Transition Sequence

[ROOT, had, little]$_S$ [effect, on, financial, markets, .]$_B$



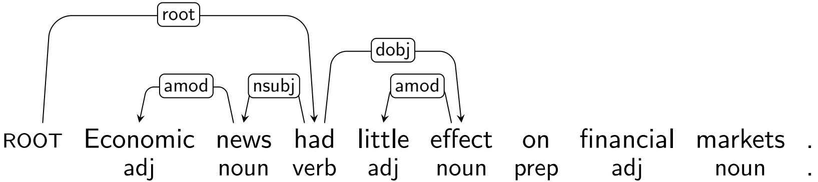
Example Transition Sequence

$[\text{ROOT, had}]_S$ $[\text{effect, on, financial, markets, .}]_B$



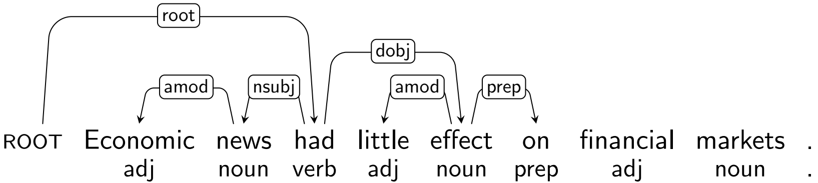
Example Transition Sequence

[ROOT, had, effect]$_S$ [on, financial, markets, .]$_B$



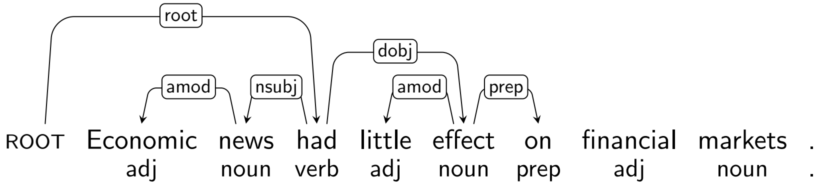
Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S$ $[\text{financial, markets, .}]_B$



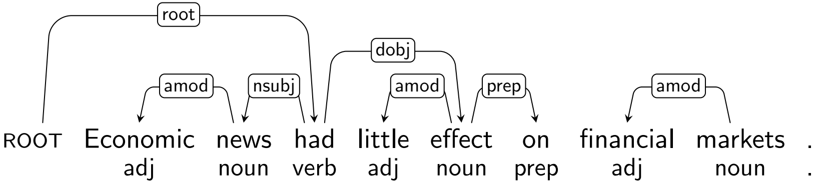
Example Transition Sequence

[ROOT, had, effect, on, financial]$_S$ [markets, .]$_B$



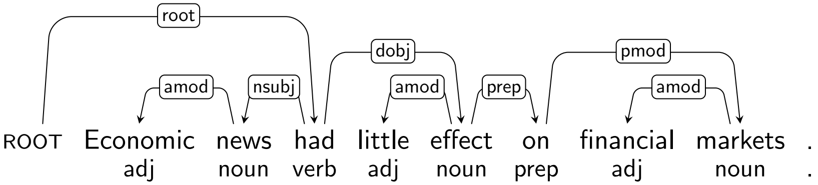
Example Transition Sequence

[ROOT, had, effect, on]$_S$ [markets, .]$_B$



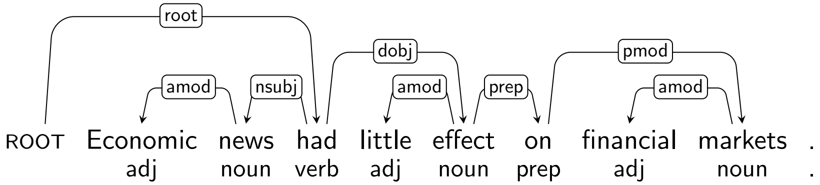
Example Transition Sequence

$[\text{ROOT, had, effect, on, markets}]_S$ $[.]_B$



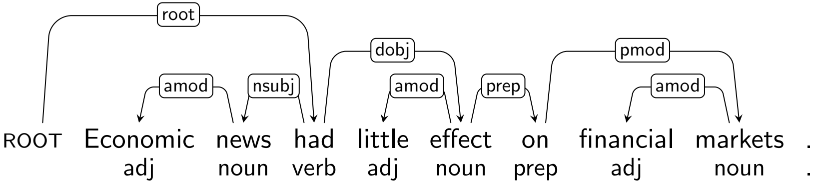
Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S \quad [.]_B$



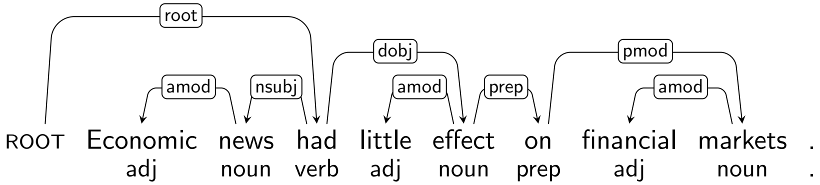
Example Transition Sequence

$[\text{ROOT, had, effect}]_S$ $[.]_B$



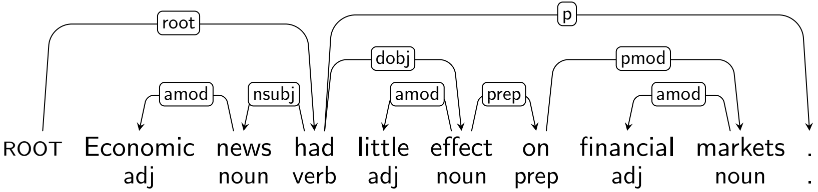
Example Transition Sequence

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



Example Transition Sequence

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



What Just Happened

SHIFT LEFT-ARC(AMOD)

SHIFT LEFT-ARC(NSUBJ)

RIGHT-ARC(ROOT)

SHIFT LEFT-ARC(AMOD)

RIGHT-ARC(DOBJ)

RIGHT-ARC(PREP)

SHIFT LEFT-ARC(AMOD)

RIGHT-ARC(PMOD)

REDUCE REDUCE REDUCE

RIGHT-ARC(P)

REDUCE REDUCE

Navigating the Parser Search Space

The Search Space

- Transition system ensures formal wellformedness of dependency trees;
- A specific sequence of transitions determines the final parsing result.

Navigating the Parser Search Space

The Search Space

- Transition system ensures formal wellformedness of dependency trees;
- A specific sequence of transitions determines the final parsing result.

Towards a Parsing Algorithm

- Abstract goal: Find transition sequence that yields the 'correct' tree.
- Learn from treebanks: output dependency tree with high probability.
- Probability distributions over transitions sequences (rather than trees).

A variant: the arc standard algorithm

- An earlier formulation of the arc eager algorithm with some limitations
- Only three transitions

SHIFT	move from front of buffer to top of stack
LEFT-ARC(K)	leftward dependency of type k between two top tokens on stack; remove 2nd token
RIGHT-ARC(K)	rightward dependency of type k between two top tokens on stack; remove top token

- Main difference: RIGHT-ARC cannot be applied until the dependent has found all its dependents

Oracle parsing

- How does the parser locate the sequence of transitions?
- Given an oracle **o** that correctly predicts the next transition $o(c)$, parsing is deterministic:

$$
\begin{array}{l}
\text{Parse}(w_1, \ldots, w_n) \\
\quad c \leftarrow ([\ ]_S, [0, 1, \ldots, n]_B, \{\ \}) \\
\quad \textbf{while } B_c \neq [\ ] \\
\qquad t \leftarrow o(c) \\
\qquad c \leftarrow t(c) \\
\quad \textbf{return } G = (\{0, 1, \ldots, n\}, A_c)
\end{array}
$$

From Oracles to Classifiers

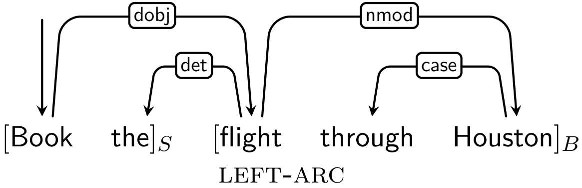
- An oracle can be approximated by a (linear) classifier:

$$o(c) = \operatorname*{argmax}_{t} \mathbf{w} \cdot \mathbf{f}(c, t)$$

- History-based feature representation $\mathbf{f}(c, t)$
- Weight vector $\mathbf{w}$ learned from treebank data

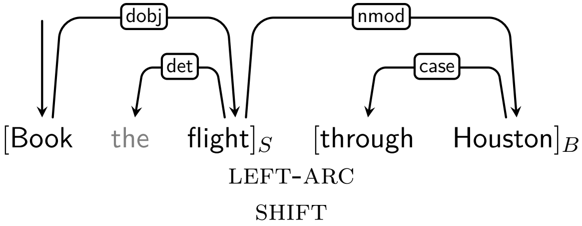
Generating training data

- Approach: simulate parsing guided by treebank data
- Given a gold standard (reference) parse and a configuration:
  - Choose LEFT-ARC if it produces a correct relation given gold
  - Choose RIGHT-ARC if it produces a correct relation given gold
  - Choose REDUCE if token is fully processed
  - Otherwise choose SHIFT



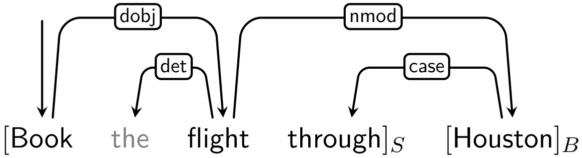
Generating training data

- Approach: simulate parsing guided by treebank data
- Given a gold standard (reference) parse and a configuration:
  - Choose LEFT-ARC if it produces a correct relation given gold
  - Choose RIGHT-ARC if it produces a correct relation given gold
  - Choose REDUCE if token is fully processed
  - Otherwise choose SHIFT



Generating training data

- Approach: simulate parsing guided by treebank data
- Given a gold standard (reference) parse and a configuration:
  - Choose LEFT-ARC if it produces a correct relation given gold
  - Choose RIGHT-ARC if it produces a correct relation given gold
  - Choose REDUCE if token is fully processed
  - Otherwise choose SHIFT



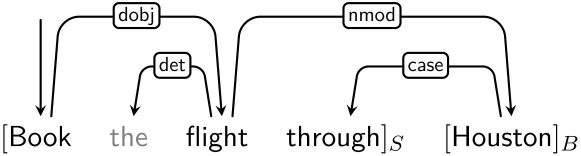
LEFT-ARC

SHIFT

SHIFT

Generating training data

- Approach: simulate parsing guided by treebank data
- Given a gold standard (reference) parse and a configuration:
  - Choose LEFT-ARC if it produces a correct relation given gold
  - Choose RIGHT-ARC if it produces a correct relation given gold
  - Choose REDUCE if token is fully processed
  - Otherwise choose SHIFT



LEFT-ARC

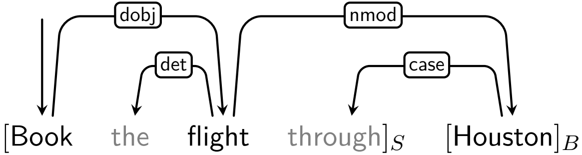
SHIFT

SHIFT

LEFT-ARC

Generating training data

- Approach: simulate parsing guided by treebank data
- Given a gold standard (reference) parse and a configuration:
  - Choose LEFT-ARC if it produces a correct relation given gold
  - Choose RIGHT-ARC if it produces a correct relation given gold
  - Choose REDUCE if token is fully processed
  - Otherwise choose SHIFT



LEFT-ARC

SHIFT

SHIFT

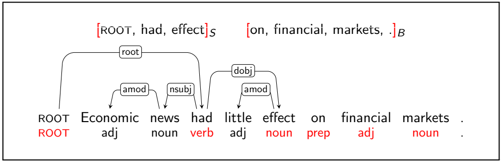
LEFT-ARC

RIGHT-ARC

Feature Representation

- Features over input tokens relative to S and B

Configuration



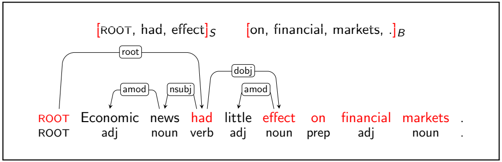
Features

$\text{pos}(S_2) = \text{ROOT}$
$\text{pos}(S_1) = \text{verb}$
$\text{pos}(S_0) = \text{noun}$
$\text{pos}(B_0) = \text{prep}$
$\text{pos}(B_1) = \text{adj}$
$\text{pos}(B_2) = \text{noun}$

Feature Representation

- Features over input tokens relative to S and B

Configuration



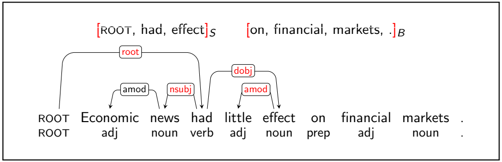
Features

$$
\begin{aligned}
\text{word}(S_2) &= \text{ROOT}\\
\text{word}(S_1) &= \text{had}\\
\text{word}(S_0) &= \text{effect}\\
\text{word}(B_0) &= \text{on}\\
\text{word}(B_1) &= \text{financial}\\
\text{word}(B_2) &= \text{markets}
\end{aligned}
$$

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A

Configuration



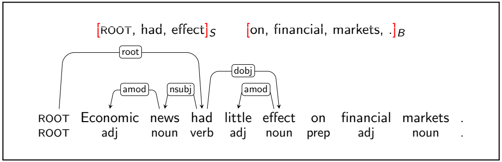
Features

$$\begin{aligned}
\text{dep}(S_1) &= \text{root} \\
\text{dep}(\text{lc}(S_1)) &= \text{nsubj} \\
\text{dep}(\text{rc}(S_1)) &= \text{dobj} \\
\text{dep}(S_0) &= \text{dobj} \\
\text{dep}(\text{lc}(S_0)) &= \text{amod} \\
\text{dep}(\text{rc}(S_0)) &= \text{NIL}
\end{aligned}$$

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A
- Features over the (partial) transition sequence

Configuration



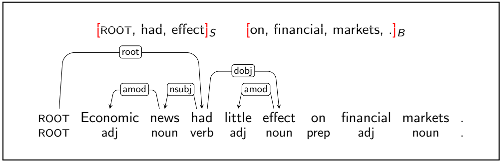
Features

t_{i-1} = Right-Arc(dobj)
t_{i-2} = Left-Arc(amod)
t_{i-3} = Shift
t_{i-4} = Right-Arc(root)
t_{i-5} = Left-Arc(nsubj)
t_{i-6} = Shift

Feature Representation

- Features over input tokens relative to S and B
- Features over the (partial) dependency graph defined by A
- Features over the (partial) transition sequence

Configuration

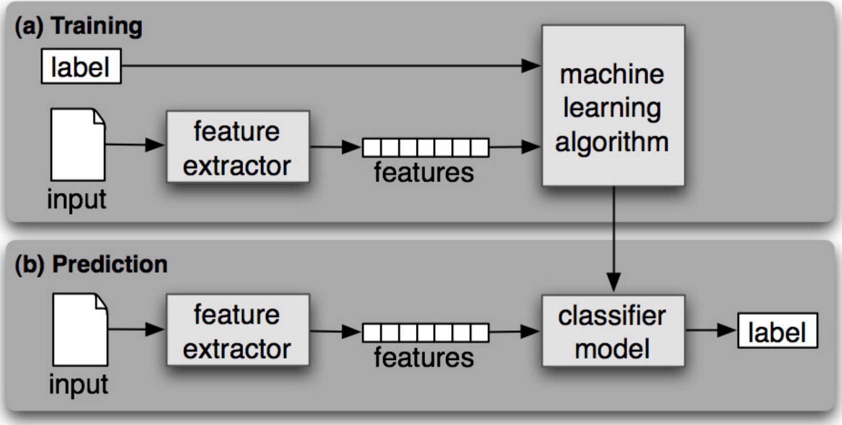


Features

$$
\begin{aligned}
t_{i-1} &= \text{Right-Arc(dobj)} \\
t_{i-2} &= \text{Left-Arc(amod)} \\
t_{i-3} &= \text{Shift} \\
t_{i-4} &= \text{Right-Arc(root)} \\
t_{i-5} &= \text{Left-Arc(nsubj)} \\
t_{i-6} &= \text{Shift}
\end{aligned}
$$

- Feature representation unconstrained by parsing algorithm

Architecture Summary



Data-driven dependency parsers

- A number of freely available dependency parsers
- Pre-trained models and trainable for any language (given available training data)
  - Stanford CoreNLP (English)
  - SpaCy (A number of languages)
  - Google SyntaxNet
  - UDParse
  - Stanza
  - etc.

Topics for Today

- Short recap:
  - Dependency syntax
  - Formal properties of dependency graphs
  - Universal Dependencies
- Syntactic parsing
- Data-driven dependency parsing
  - Variations on shift–reduce parsing
  - The arc-eager transition system
  - Thorough walk-through example
- Dependency Parser Evaluation
- Obligatory exercise

Evaluation Metrics

General Ideas

- Fixed number of tokens: per-token accuracy scores (like in tagging).

Evaluation Metrics

General Ideas

- Fixed number of tokens: per-token accuracy scores (like in tagging).
- Can consider just structure or structure plus dependency types.

General Ideas

- Fixed number of tokens: per-token accuracy scores (like in tagging).
- Can consider just structure or structure plus dependency types.
- Punctuation tokens (e.g. by Unicode property) are often excluded.

Evaluation Metrics

General Ideas

- Fixed number of tokens: per-token accuracy scores (like in tagging).
- Can consider just structure or structure plus dependency types.
- Punctuation tokens (e.g. by Unicode property) are often excluded.

UAS: Unlabeled Attachment Score

- For each token, does it have correct head (source of incoming edge)?

Evaluation Metrics

General Ideas

- Fixed number of tokens: per-token accuracy scores (like in tagging).
- Can consider just structure or structure plus dependency types.
- Punctuation tokens (e.g. by Unicode property) are often excluded.

UAS: Unlabeled Attachment Score

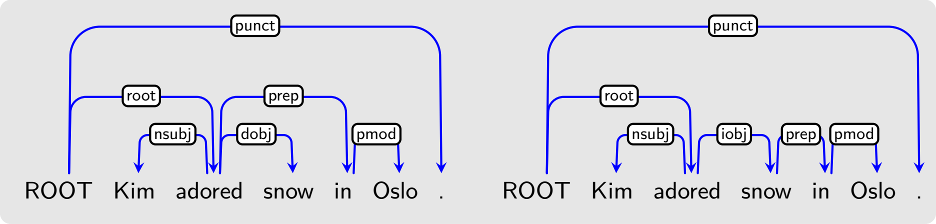
- For each token, does it have correct head (source of incoming edge)?

LAS: Labeled Attachment Score

- In addition to the head, is the dependency type (edge label) correct?

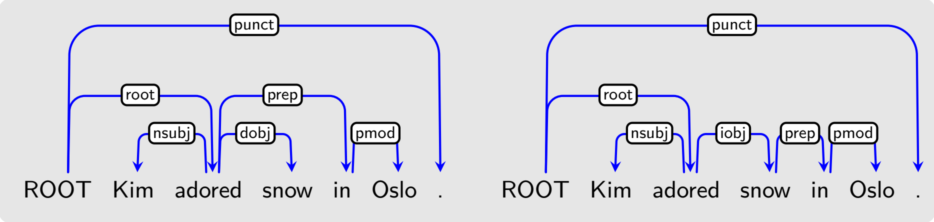
Dependency Evaluation

Gold vs. system:



Dependency Evaluation

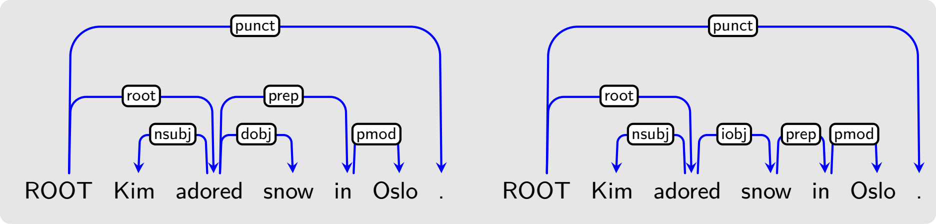
Gold vs. system:



- UAS: $4/5 = 0{,}8$

Dependency Evaluation

Gold vs. system:



- UAS: $4/5 = 0{,}8$
- LAS: $3/5 = 0{,}6$

Obligatory assignment 2a: Dependency parsing

- CoNLL-U data format
- Parsing algorithm (arc standard or arc eager)
- Train and evaluate a Norwegian dependency parser using spaCy
  - implement (unlabeled and labeled) attachment score metric
  - assess parser performance on other variants of Norwegian
- Due: April 27th 23:59

In Conclusion

Data-Driven Dependency Parsing

- No notion of grammaticality (no rules): more or less probable trees.
- Much room for experimentation: Feature models and types of classifiers;
- decent results with Maximum Entropy or Support Vector Machines.

In Conclusion

Data-Driven Dependency Parsing

- No notion of grammaticality (no rules): more or less probable trees.
- Much room for experimentation: Feature models and types of classifiers;
- decent results with Maximum Entropy or Support Vector Machines.
- In recent years, further advances with deep neural network classifiers.

In Conclusion

Data-Driven Dependency Parsing

- No notion of grammaticality (no rules): more or less probable trees.
- Much room for experimentation: Feature models and types of classifiers;
- decent results with Maximum Entropy or Support Vector Machines.
- In recent years, further advances with deep neural network classifiers.

Variants on Data-Driven Dependency Parsing

- Other transition systems (e.g. arc-standard; like 'classic' shift-reduce);
- different techniques for non-projective trees; e.g. swap transitions;
- can relax transition system further, to output general, non-tree graphs.