Dependency parsing

Benoît Sagot

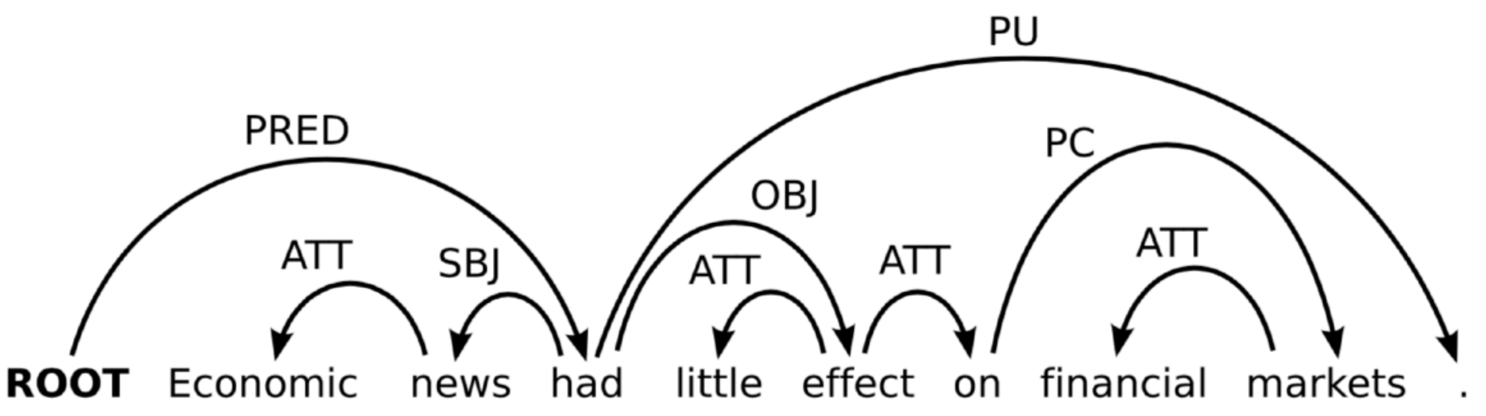
Inria (ALMAnaCH)

MVA — Speech and Language Processing — Class #6 — 24th February, 2020

Credit and disclaimer: some of the following slides are taken from, illustrated or inspired by presentations and article figures by Nivre, Dyer, Ballesteros, Kutuzov, Mooney, Rasooli and Tetreault

Introduction

- **Syntactic parsing of natural language**
  - Building the structure of natural language sentences
- **Dependency-based syntactic representations**
  - Long tradition in descriptive and theoretical linguistics
  - Have become popular in computational linguistics



Strategies for dependency parsing

- Graph-based parsing
- Transition-based parsing
- Other strategies

Graph-based parsing

- MSTParser (McDonald et al. 2005)
  - http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html
- Simplified version of the underlying idea:
  - Create all possible dependencies
  - Weigh them
  - Extract the optimal dependency tree
    - I.e. the tree that covers all words and minimises the overall weight of all retained dependencies

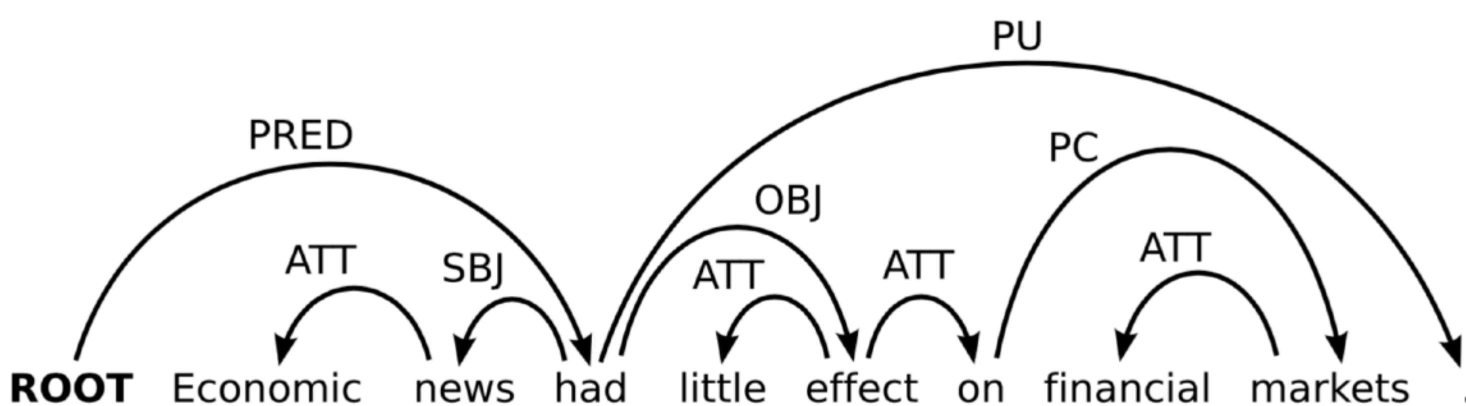
Arc-standard Transition-Based Parsing

Starting point

- **The basic idea:**
  - Define a transition system for dependency parsing
  - Learn a model for scoring possible transitions
  - Parse by searching for the optimal transition sequence
- **Advantages:**
  - Highly efficient parsing with low complexity
  - Rich history-based feature models for disambiguation
- Cf. Nivre (et al.)
  - http://www.maltparser.org

Formalising dependency trees

- A **dependency tree** is a **labelled directed tree** *T* with
  - a set *V* of nodes, labelled with wordforms (including the special “wordform” **ROOT**)
  - a set *A* of arcs, labelled with dependency types
  - a linear precedence order < on *V*
- **Notation:**
  - Arc (w_i, l, w_j) connects head w_i to dependent w_j with label l
  - Node w_0 (labeled **ROOT**) is the unique root of the tree



Parser configurations

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

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

  (recall that w_0 = **ROOT**)
- **Termination**: $([w_0]_S, []_Q, A)$

Transitions for the “arc-standard algorithm”

- **Left-Arc**(*l*)

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

- **Right-Arc**(*l*)

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

- **Shift**

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

Example Transition Sequence

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

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

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

action: Shift

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

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

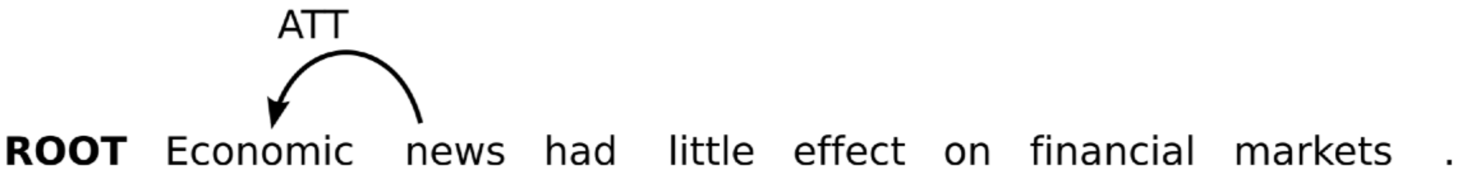
action: Shift

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

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

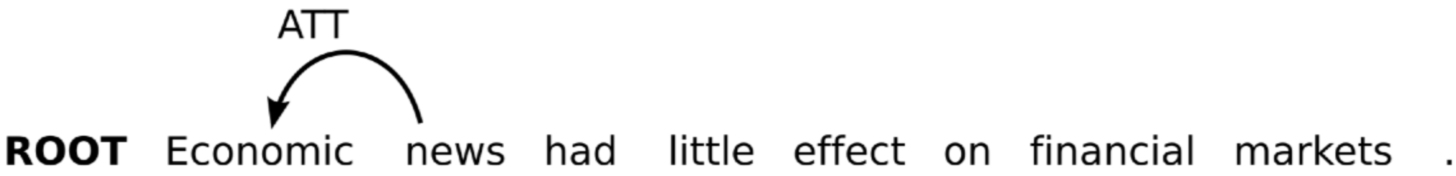
action: Left-Arc(ATT)



Example Transition Sequence

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

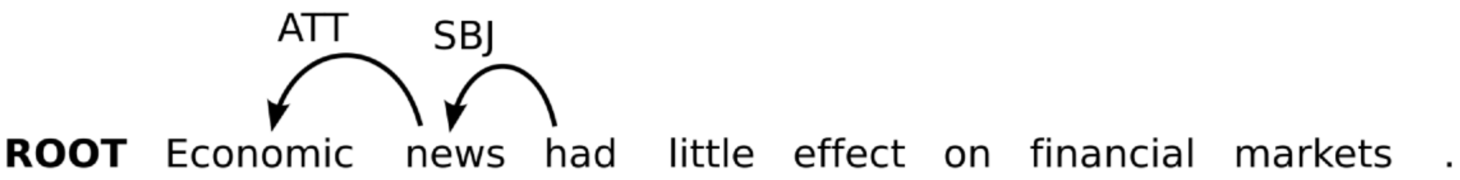
action: Shift



Example Transition Sequence

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

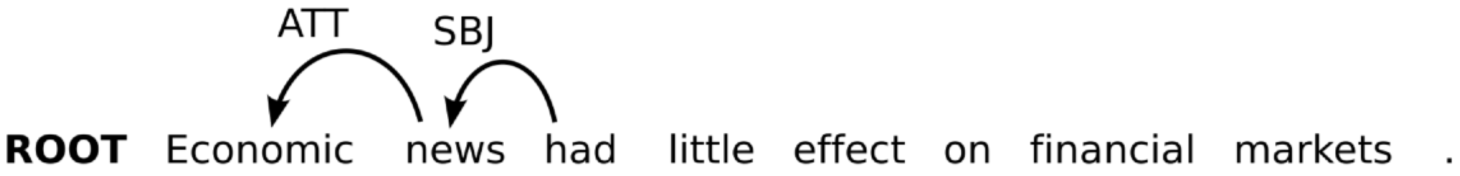
action: Left-Arc(SBJ)



Example Transition Sequence

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

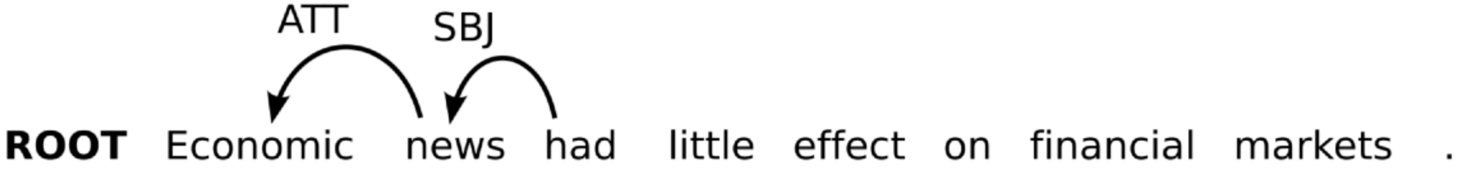
action: Shift



Example Transition Sequence

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

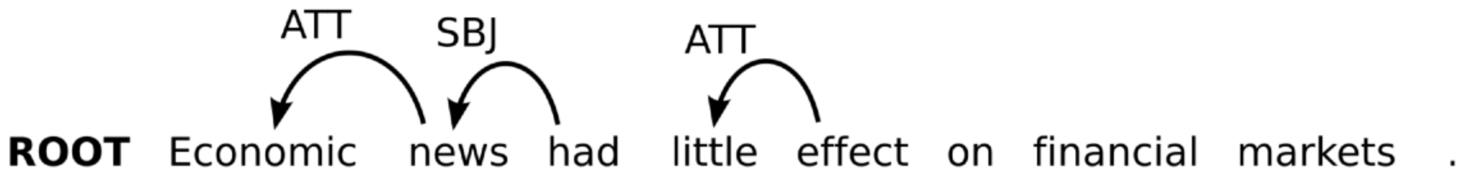
action: Shift



Example Transition Sequence

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

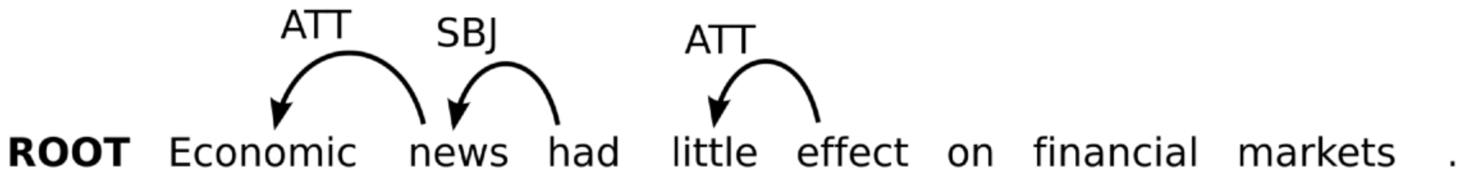
action: Left-Arc(ATT)



Example Transition Sequence

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

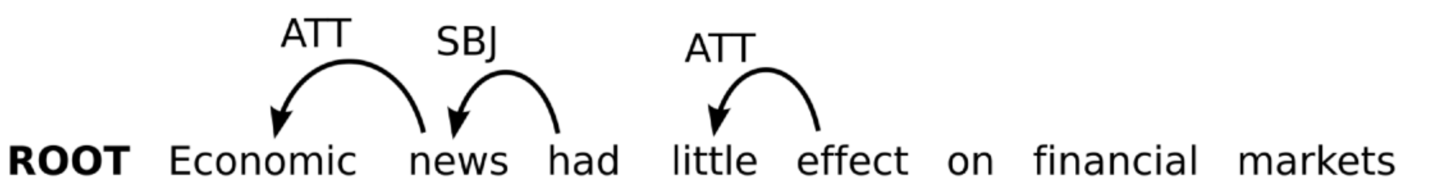
action: Shift



Example Transition Sequence

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

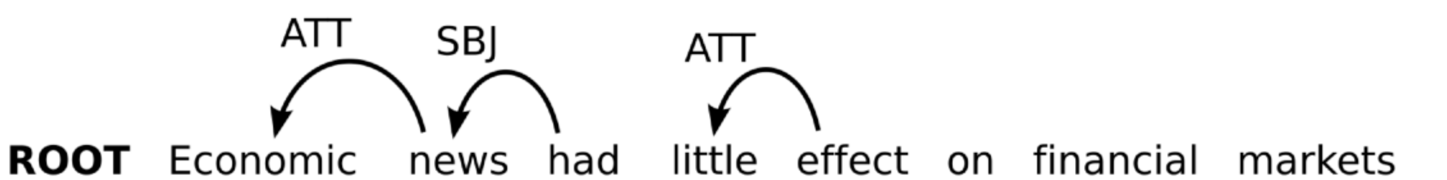
action: Shift



Example Transition Sequence

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

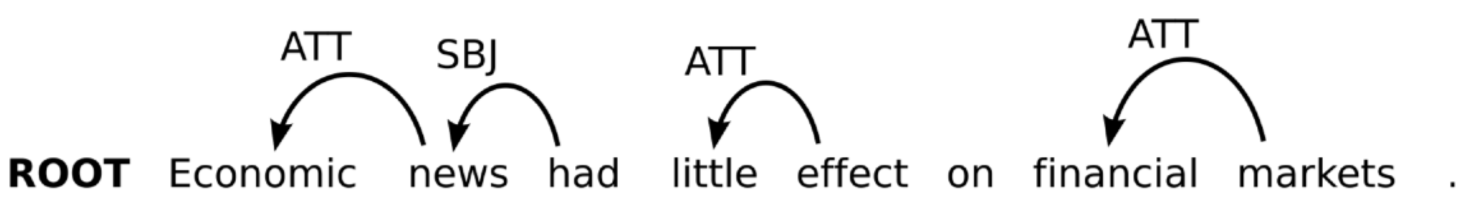
action: Shift



Example Transition Sequence

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

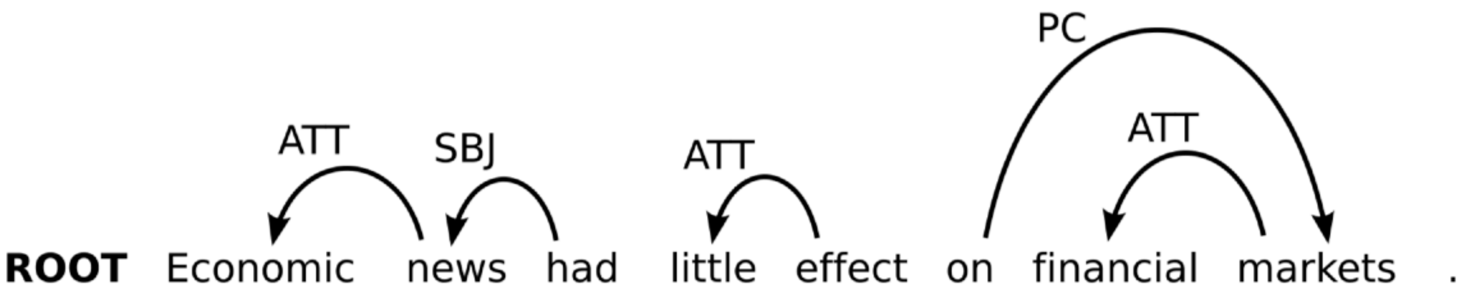
action: Left-Arc(ATT)



Example Transition Sequence

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

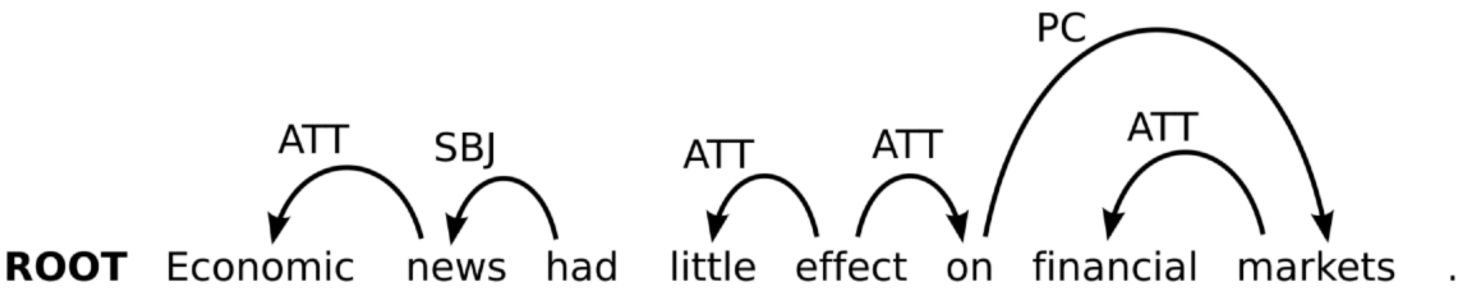
action: Right-Arc(PC)



Example Transition Sequence

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

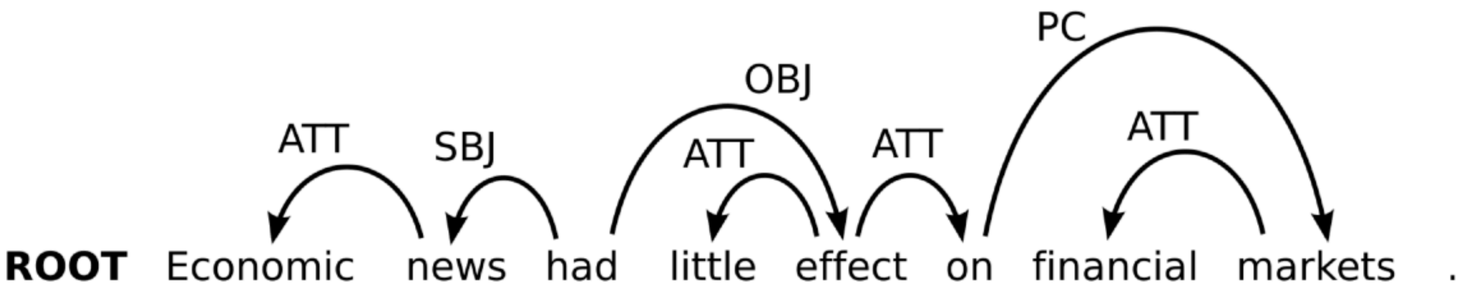
action: Right-Arc(ATT)



Example Transition Sequence

[ROOT, had, effect]$_S$ [.]$_Q$

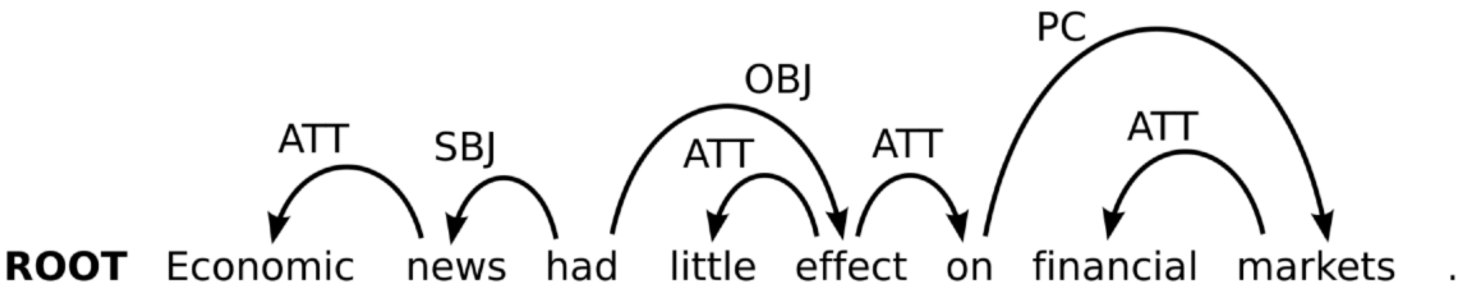
action: Right-Arc(OBJ)



Example Transition Sequence

$[\text{ROOT, had, .}]_S \; []_Q$

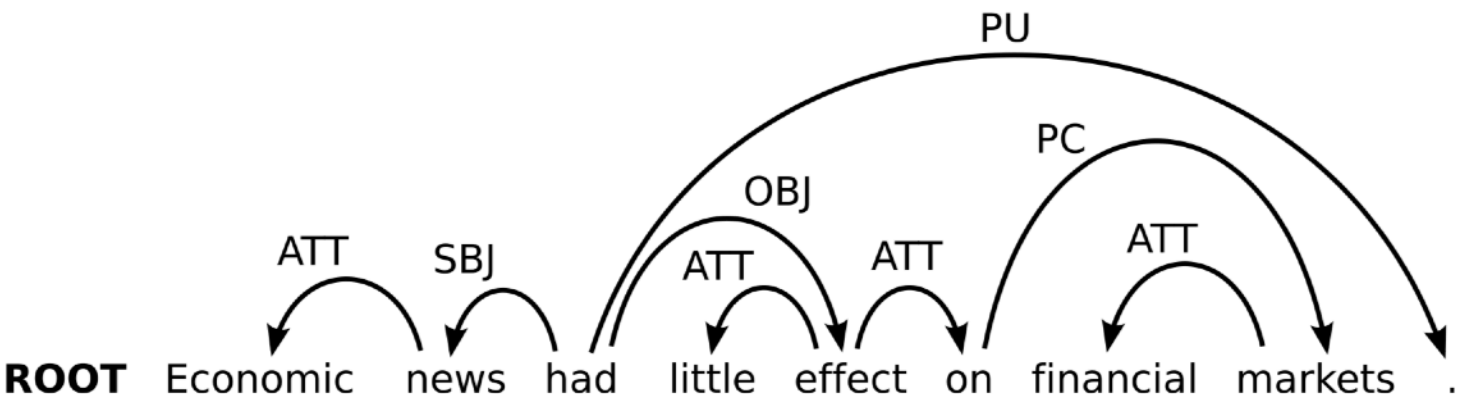
action: Shift



Example Transition Sequence

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

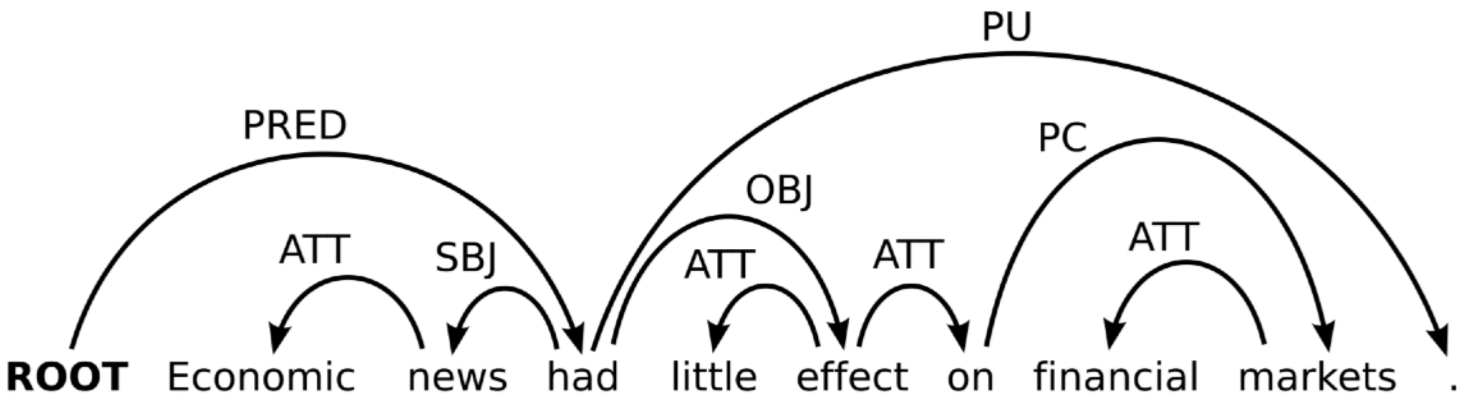
action: Right-Arc(PU)



Example Transition Sequence

[ROOT, had]$_S$ []$_Q$

action: Right-Arc(PRED)



Properties of the algorithm

- Every transition sequence outputs a projective dependency tree (**soundness**).
- Every projective dependency tree is output by some transition sequence (**completeness**).
- There are exactly 2*n* transitions in a sentence with *n* words.

Deterministic parsing

- If we have an **oracle** that correctly predicts the next transition *o(c)*, parsing is deterministic:

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

Oracles as classifiers

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

$$o(c) = \text{argmax}_t\ \mathbf{w} \cdot \mathbf{f}(c, t)$$

- History-based feature representation $\mathbf{f}(c, t)$:
  - Features over input tokens relative to *S* and *Q*
  - Features over the (partial) dependency tree defined by *A*
  - Features over the (partial) transition sequence
- Weight vector **w** learned from treebank data:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set $D = \{(c, t) \mid o(c) = t\}$
  - Maximise accuracy of local predictions $o(c) = t$

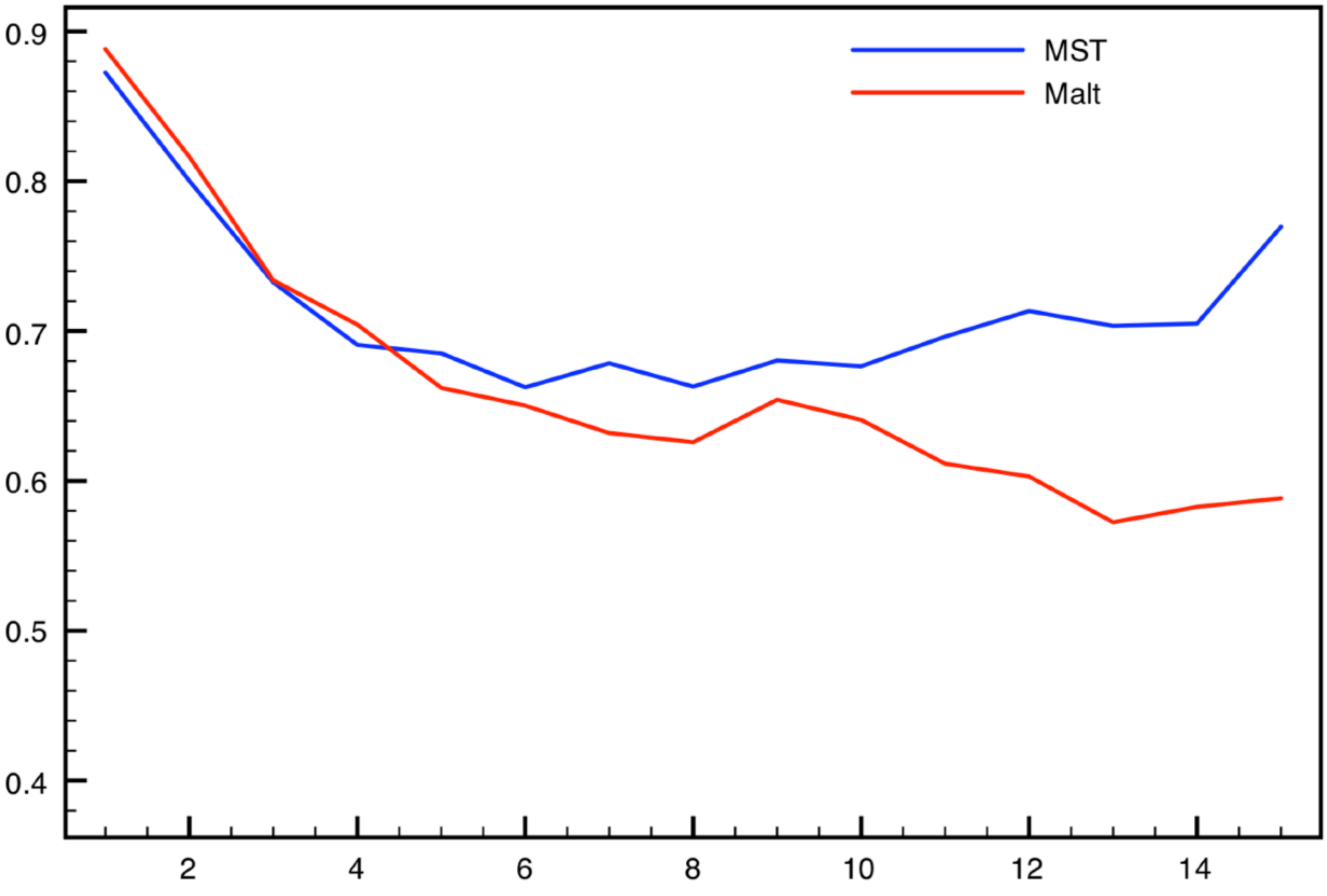
Deterministic classifier-based parsing

- 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 deterministic parsing and local learning

Empirical results: the CoNLL 2006 shared task

- CoNLL 2006 shared task (Buchholz and Marsi 2006):
  - **MaltParser** (Nivre et al. 2006) – transition-based, deterministic, local learning
  - **MSTParser** (McDonald et al. 2006) – graph-based, exact, global learning
  - Same average parsing accuracy over 13 languages
- Comparative **error analysis** (McDonald and Nivre2007):
  - MaltParser more accurate on short dependencies and disambiguation of core grammatical functions
  - MSTParser more accurate on long dependencies and dependencies near the root of the tree
- Hypothesised **explanation for MaltParser results**:
  - **Rich features counteracted by error propagation**

Precision by dependency length



Beam search
and structured prediction

Beam search

- **Maintain the *k* best hypotheses** (Johansson and Nugues 2006):

$\textsc{Parse}(w_1, \ldots, w_n)$

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

$\quad \textbf{while } \exists c \in \textsc{Beam}\ [Q_c \neq [\,]\ \textbf{or}\ |S_c| > 1]$

$\quad\quad \textbf{foreach } c \in \textsc{Beam}$

$\quad\quad\quad \textbf{foreach } t$

$\quad\quad\quad\quad \textsc{Add}(t(c), \textsc{NewBeam})$

$\quad\quad \textsc{Beam} \leftarrow \textsc{Top}(k, \textsc{NewBeam})$

$\quad \textbf{return } T = (\{w_0, w_1, \ldots, w_n\}, A_{\textsc{Top}(1,\, \textsc{Beam})})$

- **Note:**
  - Score(c_0, … , c_m) = $\sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{f}(c_{j-1}, t_j)$
  - Simple combination of locally normalised classifier scores
  - Marginal gains in accuracy

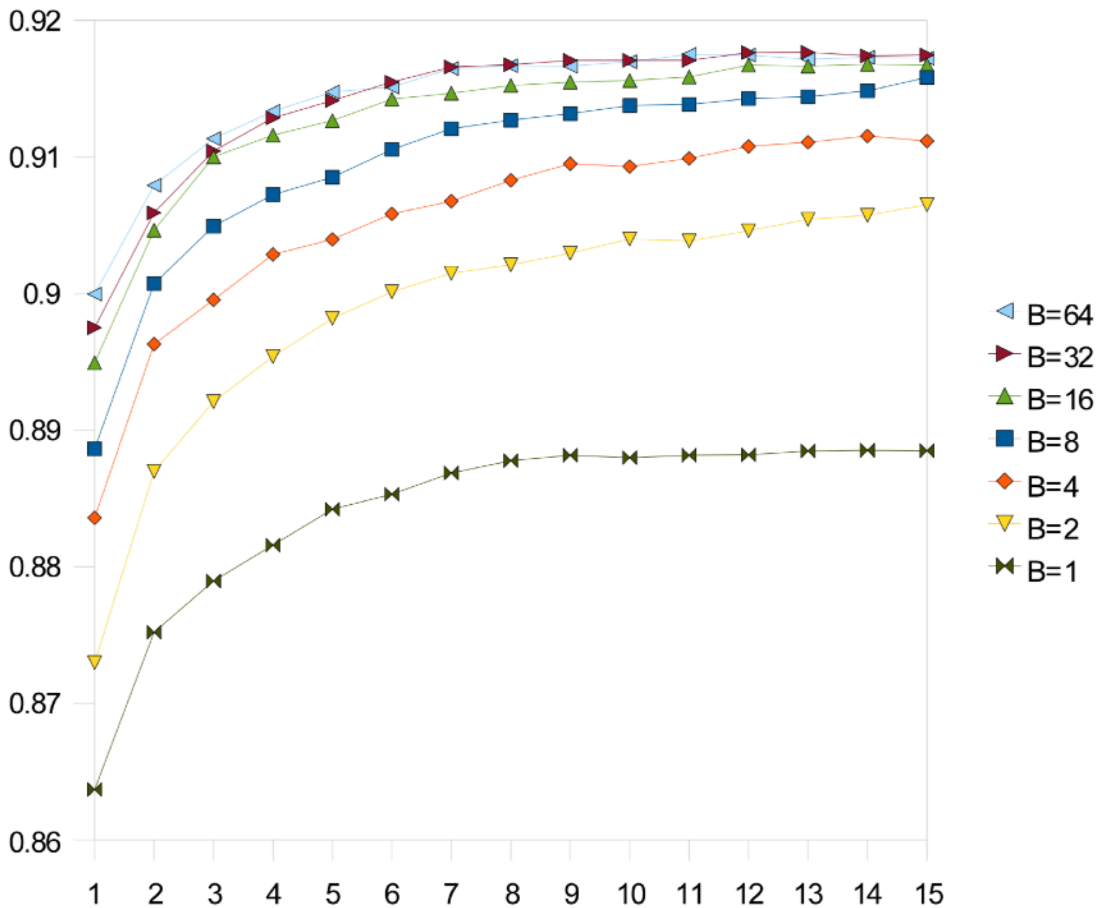
Structured prediction

- **Parsing as structured prediction** (Zhang and Clark 2008)**:**
  - Minimise 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 updates (Collins and Roark 2004)

Beam size and training iterations

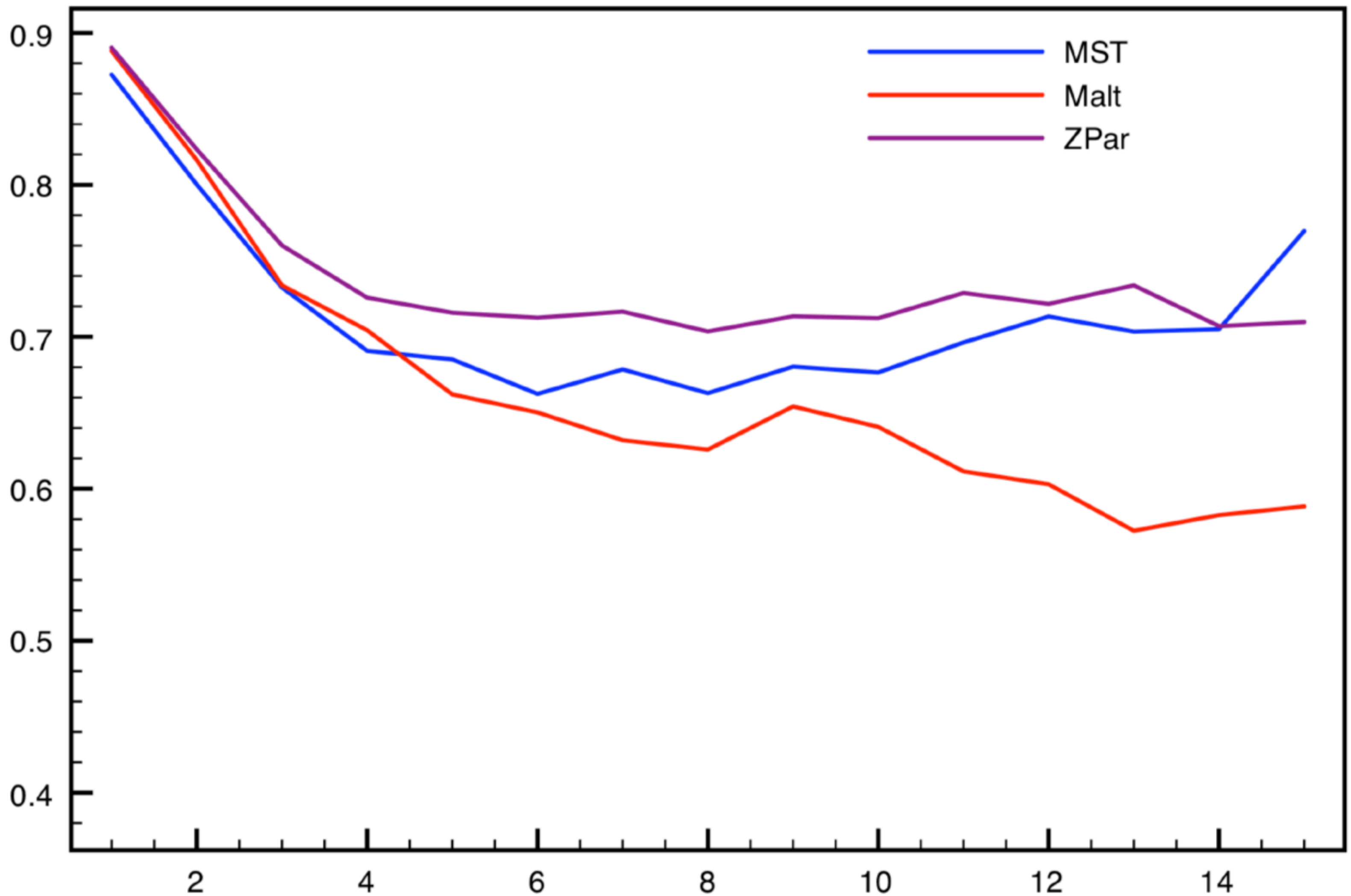


Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-Based and Transition-Based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 562–571.

The best of two worlds?

- Like graph-based dependency parsing (MSTParser):
  - Global learning – minimise loss over entire sentence
  - Non-greedy search – accuracy increases with beam size
- Like deterministic transition-based parsing (MaltParser):
  - Highly efficient – complexity still linear for fixed beam size
  - Rich features – no constraints from parsing algorithm
- Example ZPar parser (Zhang and Clark 2011)
  - “Most heavily developed for English and Chinese”

Precision by dependency length, again



Even richer feature models

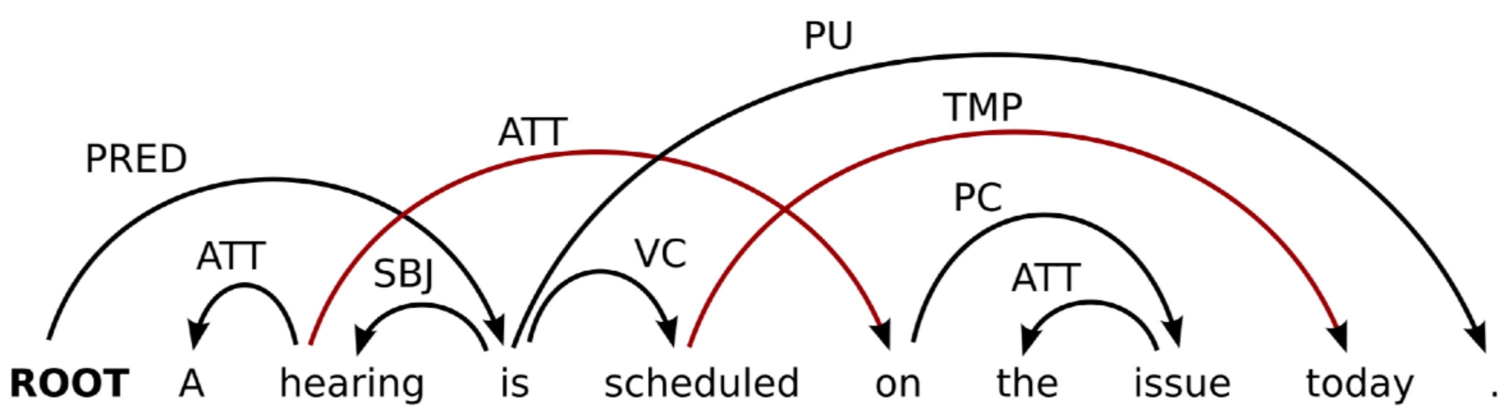
	ZPar	Malt
Baseline	92.18	89.37
+distance	+0.07	–0.14
+valency	+0.24	0.00
+unigrams	+0.40	–0.29
+third-order	+0.18	0.00
+label set	+0.07	+0.06
Extended	93.14	89.00

Yue Zhang and Joakim Nivre. 2011. Transition-Based Dependency Parsing with Rich Non-Local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 188–193.

Online reordering for non-projectivity

Projectivity

- A dependency arc is **projective** if the head (transitively) dominates all intervening words
- Most dependency grammar theories do not assume projectivity (but many parsers do)

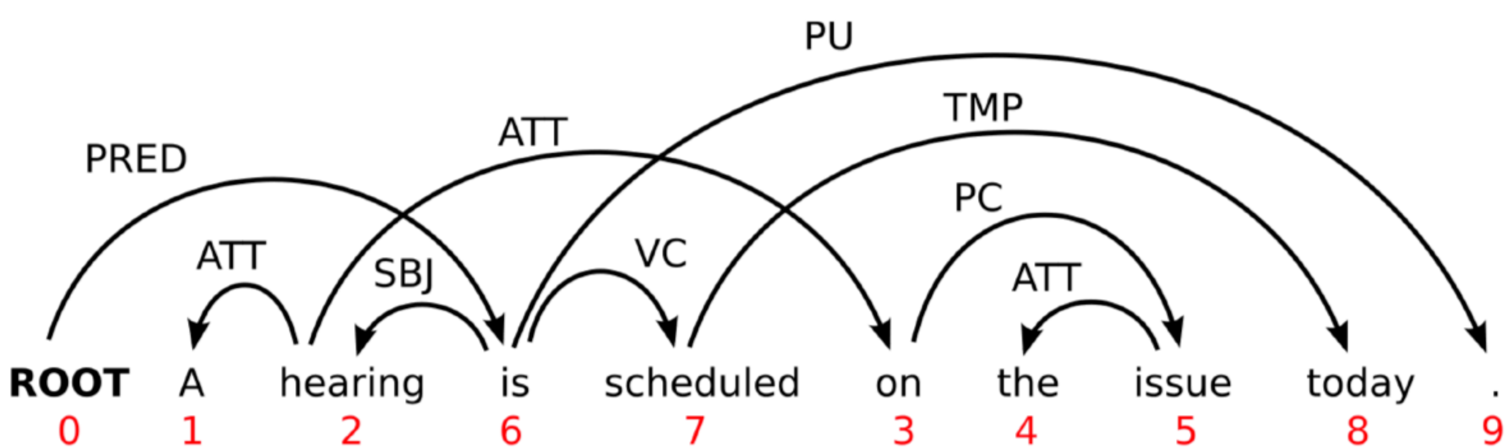


Non-projectivity in natural languages

Language	Trees	Arcs
Arabic (Hajič et al. 2004)	11,2 %	0,4 %
Basque (Aduriz et al. 2003)	26,2 %	2,9 %
Czech (Hajič et al. 2001)	23,2 %	1,9 %
Danish (Kromann 2003)	15,6 %	1,0 %
Greek (Prokopidis et al. 2005)	20,3 %	1,1 %
Russian (Boguslavsky et al. 2000)	10,6 %	0,9 %
Slovene (Džeroski et al. 2006)	22,2 %	1,9 %
Turkish (Oflazer et al. 2003)	11,6 %	1,5 %

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 in-order traversal of T with respect to $<$ (Veselá et al. 2004)



Parsing with online reordering

- Add transition for reordering words (Nivre 2009):
  - **Swap**

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

- Transition-based parsing with two interleaved processes:
  - Sort words into projective order $<_p$
  - Build dependency tree T by connecting adjacent subtrees
    - *T* is always projective with respect to $<_p$
    - *T* may be non-projective with respect to $<$

Example Transition Sequence

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

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

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

action: Shift

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

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

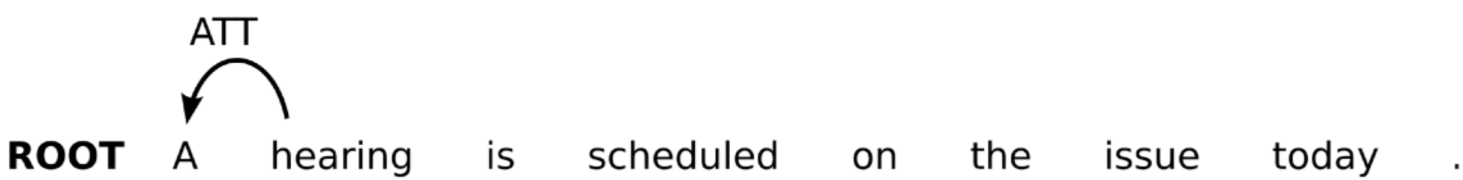
action: Shift

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

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

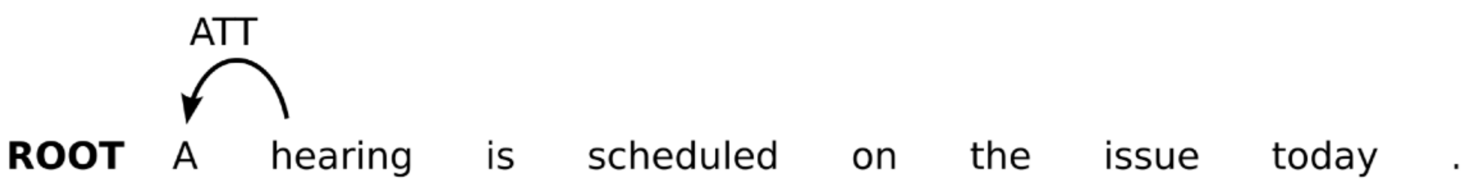
action: Left-Arc(ATT)



Example Transition Sequence

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

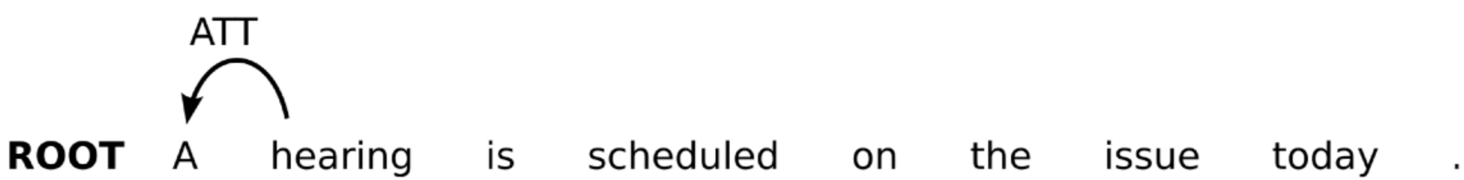
action: Shift



Example Transition Sequence

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

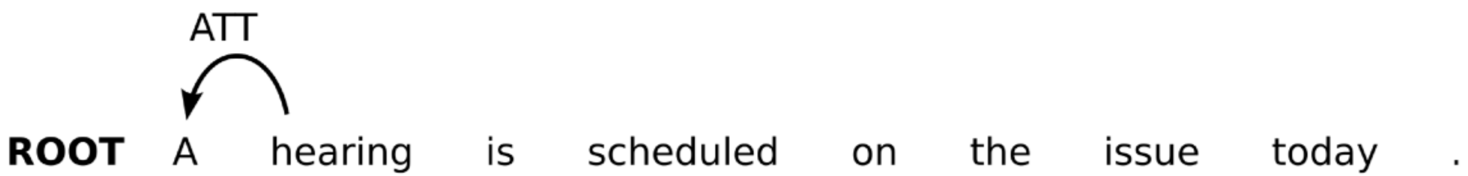
action: Shift



Example Transition Sequence

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

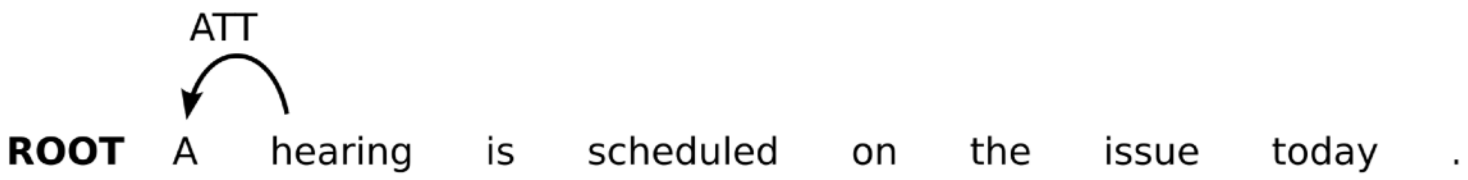
action: Shift



Example Transition Sequence

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

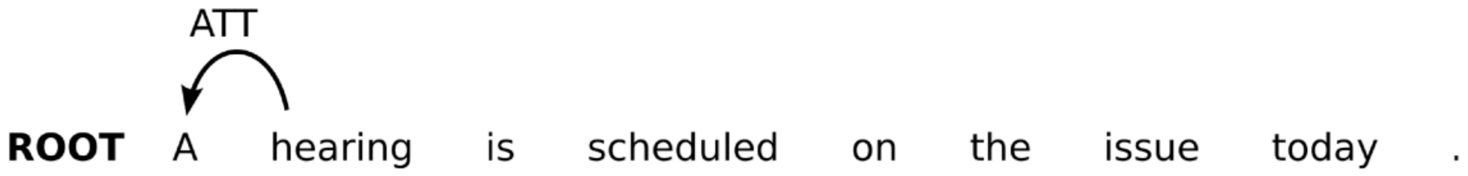
action: Shift



Example Transition Sequence

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

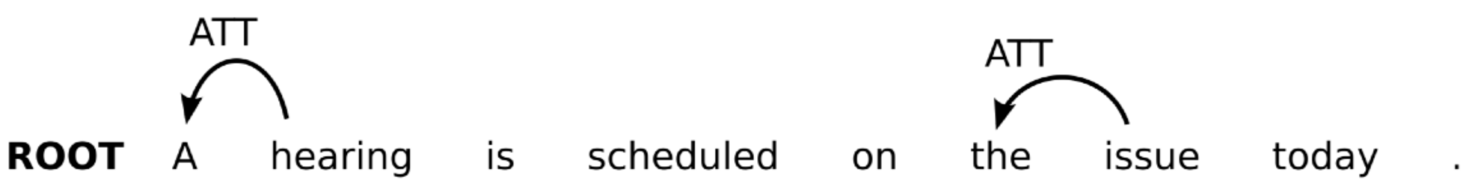
action: Shift



Example Transition Sequence

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

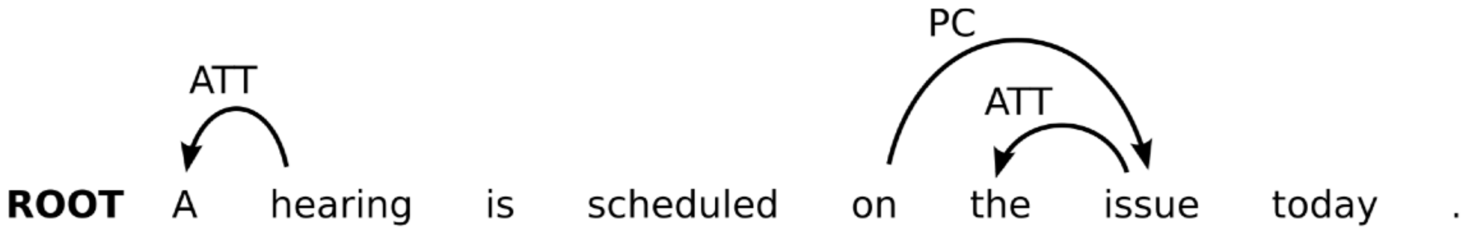
action: Left-Arc(ATT)



Example Transition Sequence

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

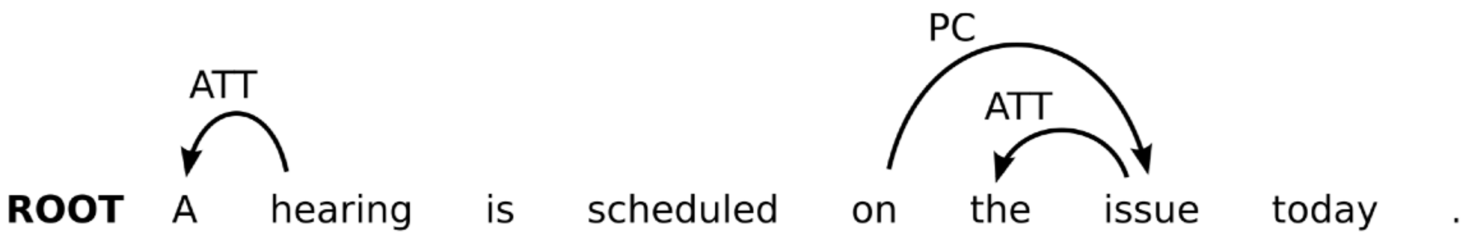
action: Right-Arc(PC)



Example Transition Sequence

[ROOT, hearing, is, on]$_S$ [scheduled, today, .]$_Q$

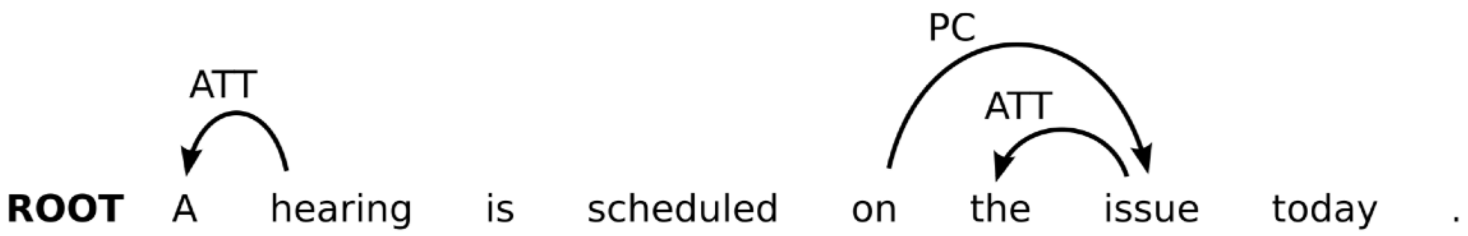
action: Swap



Example Transition Sequence

[ROOT, hearing, on]$_S$ [is, scheduled, today, .]$_Q$

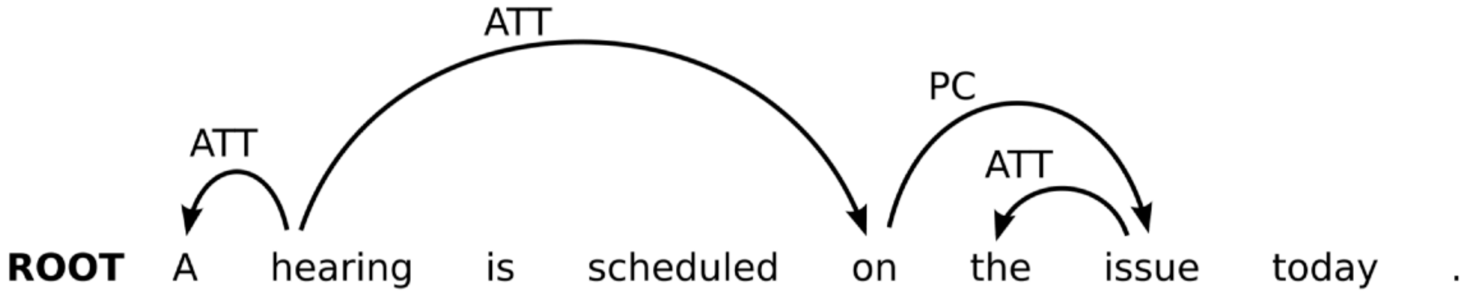
action: Swap



Example Transition Sequence

[ROOT, hearing, on]$_S$ [is, scheduled, today, .]$_Q$

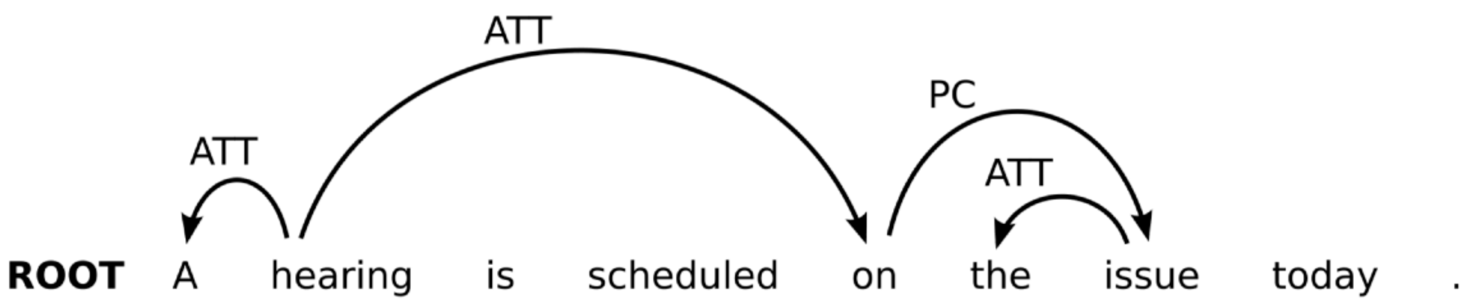
action: Right-Arc(ATT)



Example Transition Sequence

[ROOT, hearing, is]$_S$ [scheduled, today, .]$_Q$

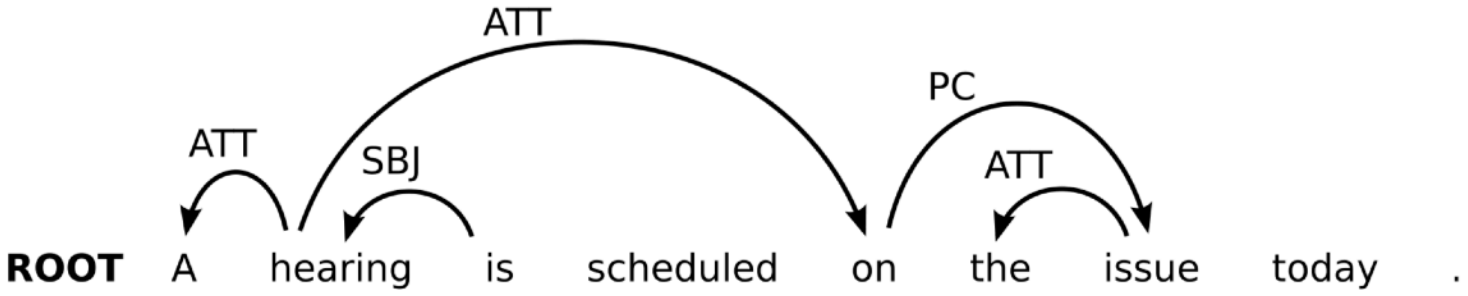
action: Shift



Example Transition Sequence

[ROOT, hearing, is]$_S$ [scheduled, today, .]$_Q$

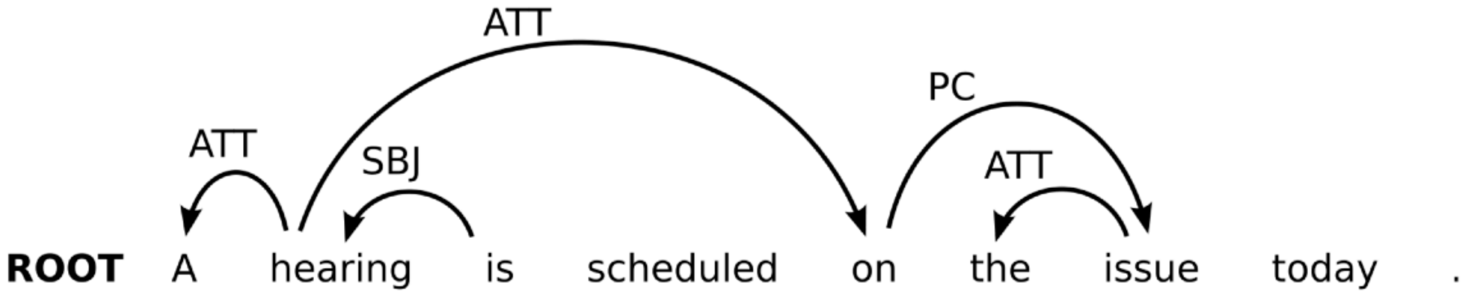
action: Left-Arc(SBJ)



Example Transition Sequence

[ROOT, is, scheduled]$_S$ [today, .]$_Q$

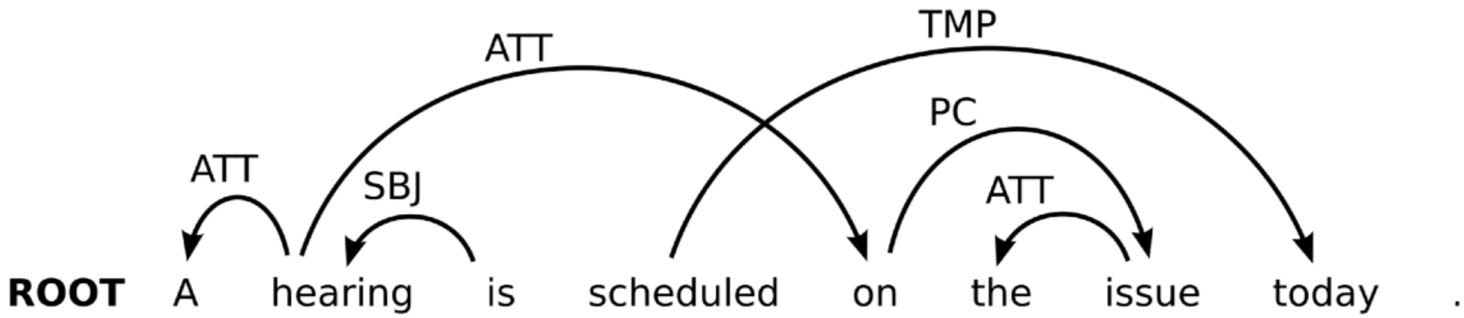
action: Shift



Example Transition Sequence

[ROOT, is, scheduled, today]$_S$ [.]$_Q$

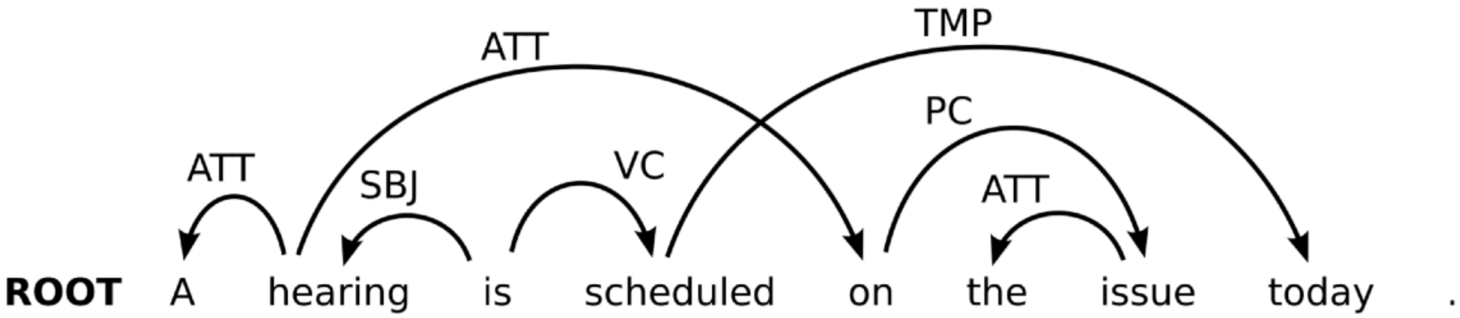
action: Right-Arc(TMP)



Example Transition Sequence

[ROOT, is, scheduled]$_S$ [.]$_Q$

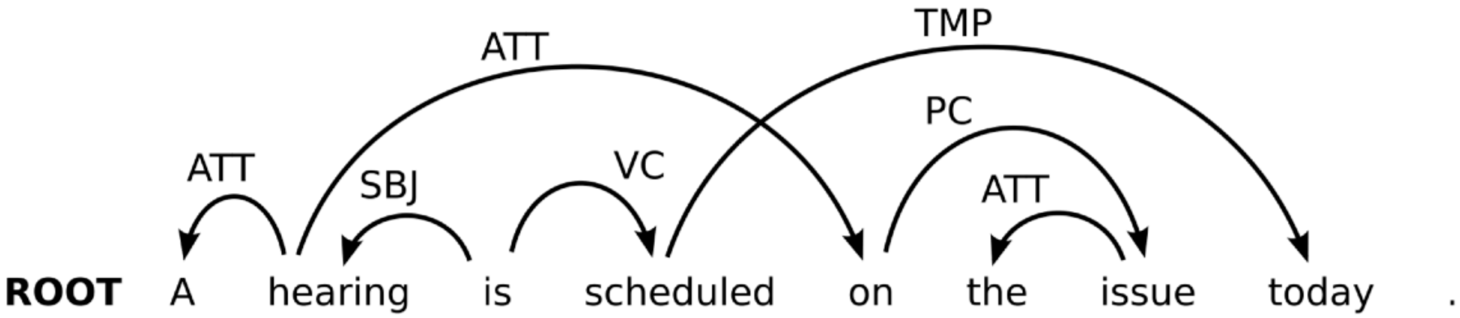
action: Right-Arc(VC)



Example Transition Sequence

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

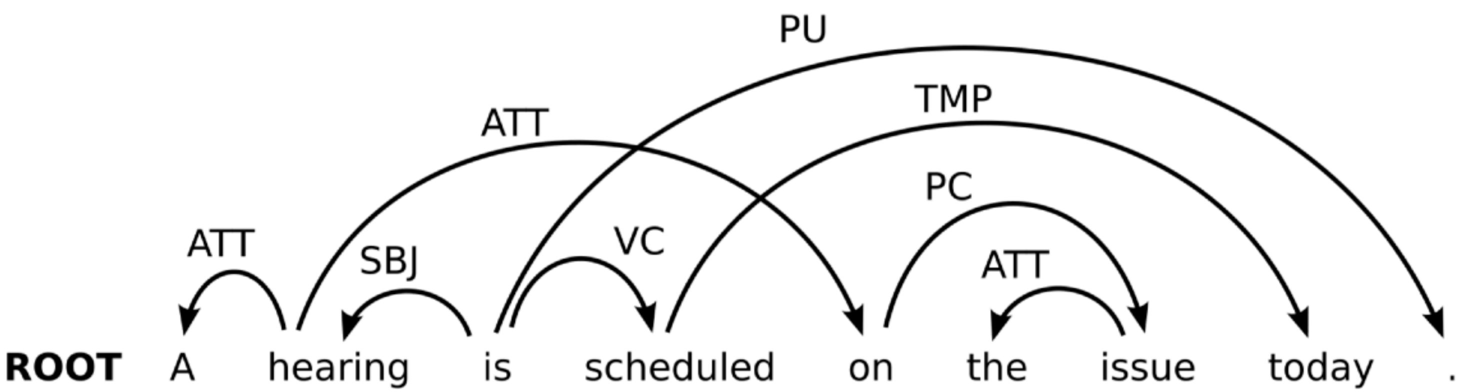
action: Shift



Example Transition Sequence

[ROOT, is, .]$_S$ []$_Q$

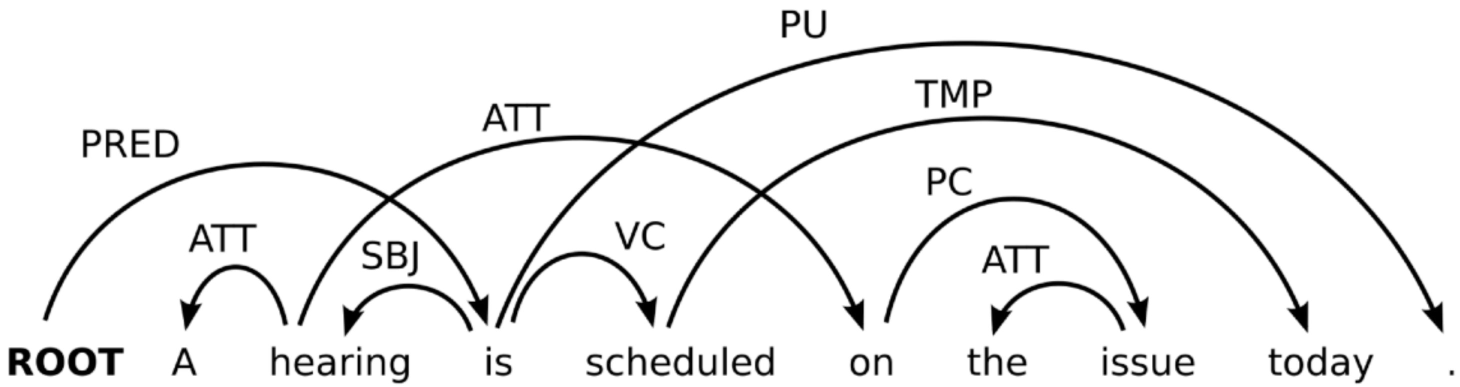
action: Right-Arc(PU)



Example Transition Sequence

[ROOT, is]$_S$ []$_Q$

action: Right-Arc(PRED)



Empirical results

- Deterministic transition-based parsing (Nivre 2009):
  - Parsing in linear expected time (quadratic worst-case time)
  - Best results on Czech CoNLL 2006 data sets
- Beam search and structured prediction:
  - Evaluation on CoNLL 2009 data sets (dev sets)

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

Arc-eager
Transition-Based Parsing

Limitations of the arc-standard algorithm

- The **arc-standard** system considered so far
  - builds a dependency tree strictly bottom-up
  - a dependency arc can only be added between two nodes if the dependent node has already found all its dependents.
  - As a consequence, it is often necessary to postpone the attachment of right dependents.
- This is a problem, as parsing decisions are easier to take when the governor and the governee of a dependency are immediately accessible

The problem of arc-standard on an example

$[\text{ROOT}]_S$ $[\text{La, température, a, un, très, gros, effet, sur, la, concentration}]_Q$

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

$[\text{ROOT, La}]_S$ $[\text{température, a, un, très, gros, effet, sur, la, concentration}]_Q$

action: Shift

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, La, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, La, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

The problem of arc-standard on an example

[ROOT, température, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, température, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

The problem of arc-standard on an example

[ROOT, a, un]$_S$ [très, gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, très, gros]$_S$ [effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, a, un, très, gros]$_S$ [effet, sur, la, concentration]$_Q$

action: Right-Arc()

The problem of arc-standard on an example

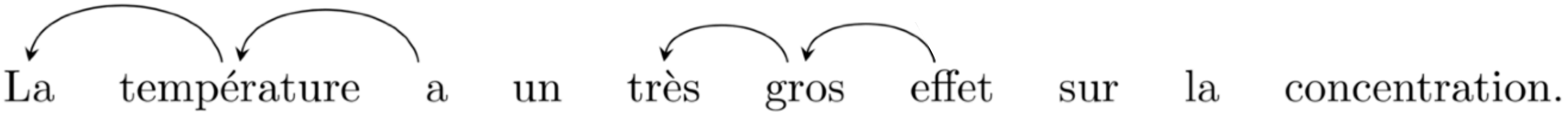
[ROOT, a, un, gros, effet]$_S$ [sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, gros, effet]$_S$ [sur, la, concentration]$_Q$

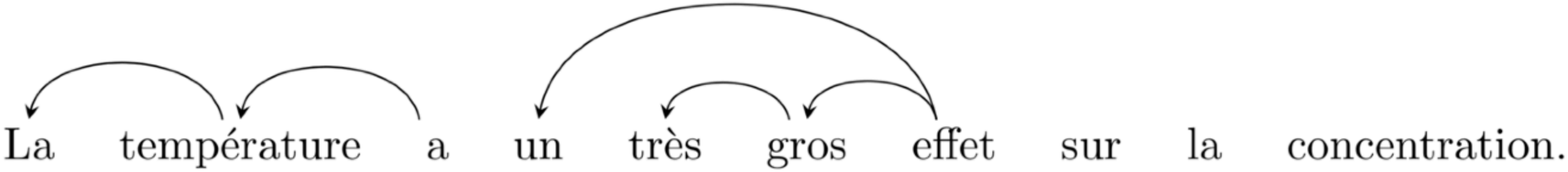
action: Left-Arc()



The problem of arc-standard on an example

[ROOT, a, un, effet]$_S$ [sur, la, concentration]$_Q$

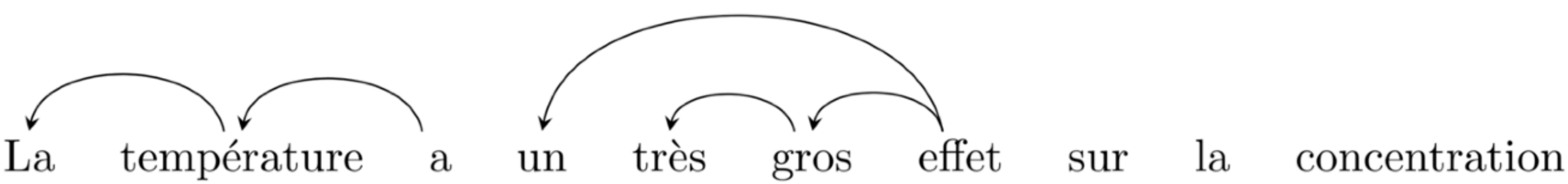
action: Left-Arc()



The problem of arc-standard on an example

[ROOT, a, effet, sur]$_S$ [la, concentration]$_Q$

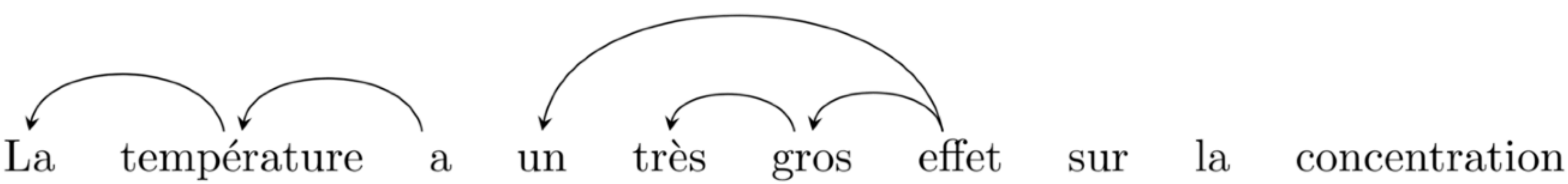
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la]$_S$ [concentration]$_Q$

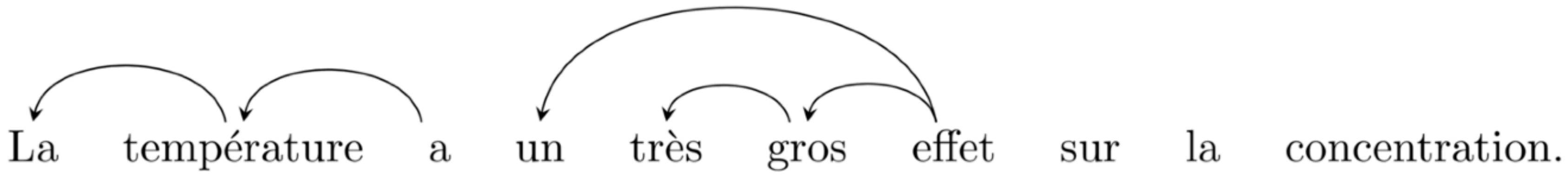
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la, concentration]$_S$ []$_Q$

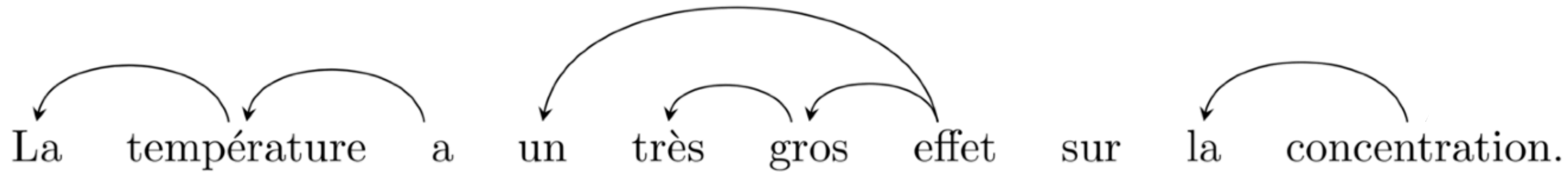
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la, concentration]$_S$ []$_Q$

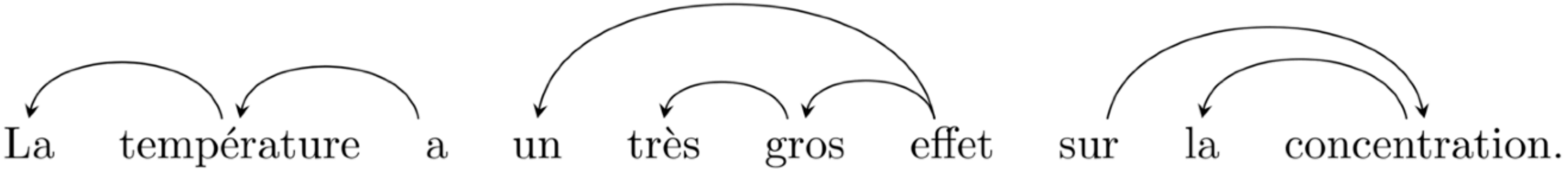
action: Left-Arc()



The problem of arc-standard on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

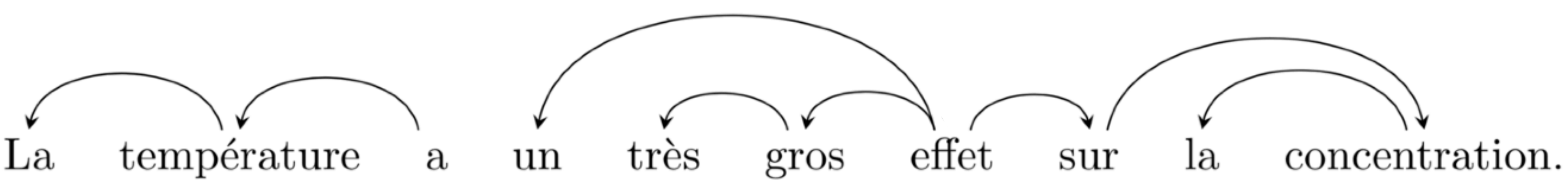
action: Right-Arc()



The problem of arc-standard on an example

[ROOT, a, effet, sur]$_S$ []$_Q$

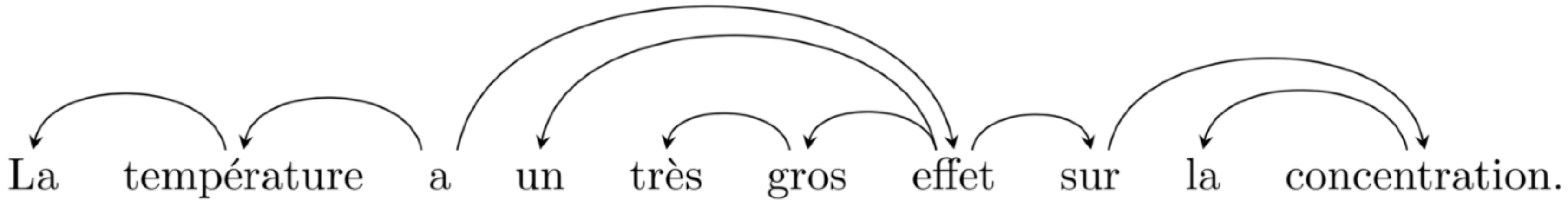
action: Right-Arc()



The problem of arc-standard on an example

$[\text{ROOT, a, effet}]_S \; []_Q$

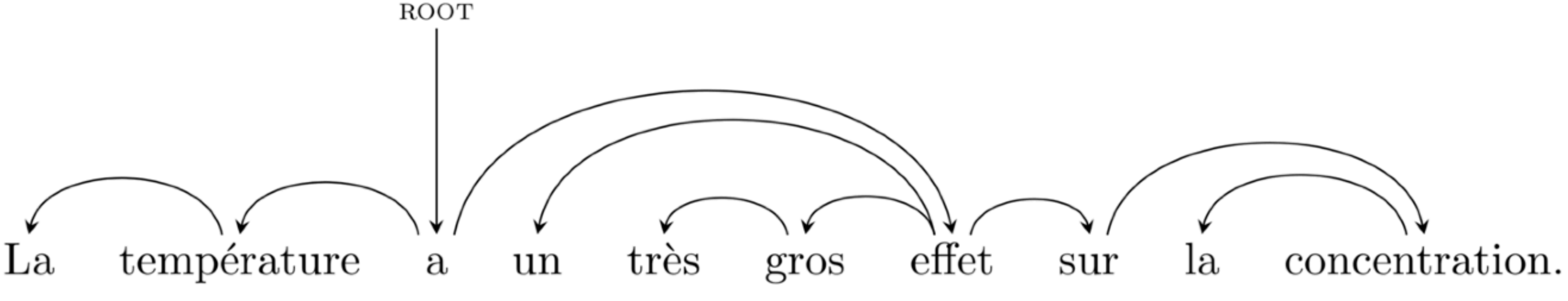
action: Right-Arc()



The problem of arc-standard on an example

[ROOT, a]$_S$ []$_Q$

action: Right-Arc()



The arc-eager system

- We will modify the basic set of actions in order to always add an arc at the earliest possible opportunity:
  - we will now build parts of the tree top-down instead of bottom-up
- Shift remains the same
- **Left-Arc is rewritten and subjected to a stricter condition** (allowed only if the dependent is not the root and has no incoming arcs)

$$\frac{([\ldots, w_i]_S, [w_j, \ldots]_Q, A)}{([\ldots]_S, [w_j, \ldots]_Q, A \cup \{(w_j, l, w_i)\})} \quad [i \neq 0 \wedge \nexists(k, l') \mid (k, l', i) \in A]$$

The arc-eager system

- **Right-Arc is changed**: it does not discard w_i anymore:

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

- We postpone the reduction of w_i to another, new action:
- **Reduction**, only possible if the top of the stack already has a head

$$\frac{([\ldots, w_i]_S, Q, A)}{([\ldots]_S, Q, A)} \text{ only if } \exists (k, l') \mid (k, l', i) \in A$$

Arc-eager on an example

$[\text{ROOT}]_S$ [La, température, a, un, très, gros, effet, sur, la, concentration]$_Q$

La température a un très gros effet sur la concentration.

Arc-eager on an example

[ROOT, La]$_S$ [température, a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

Arc-eager on an example

[ROOT, La]$_S$ [température, a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

La température a un très gros effet sur la concentration.

Arc-eager on an example

[ROOT, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

Arc-eager on an example

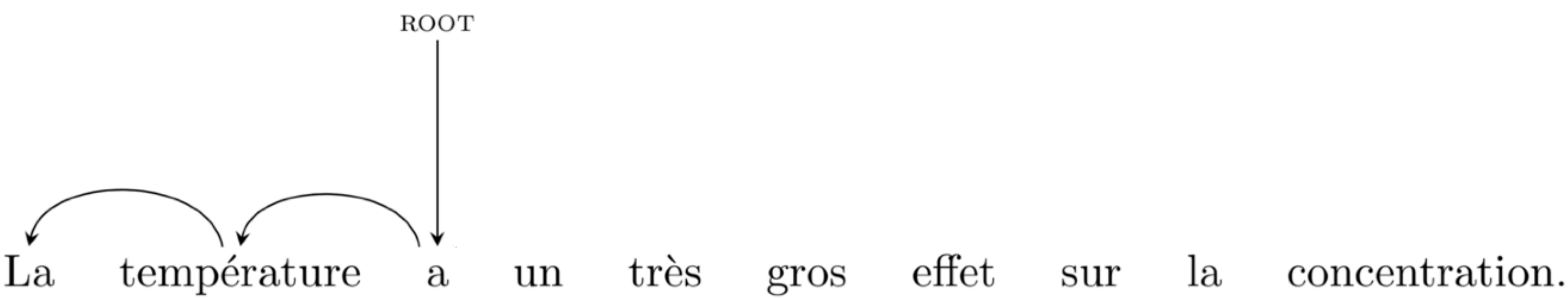
[ROOT, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

Arc-eager on an example

[ROOT, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

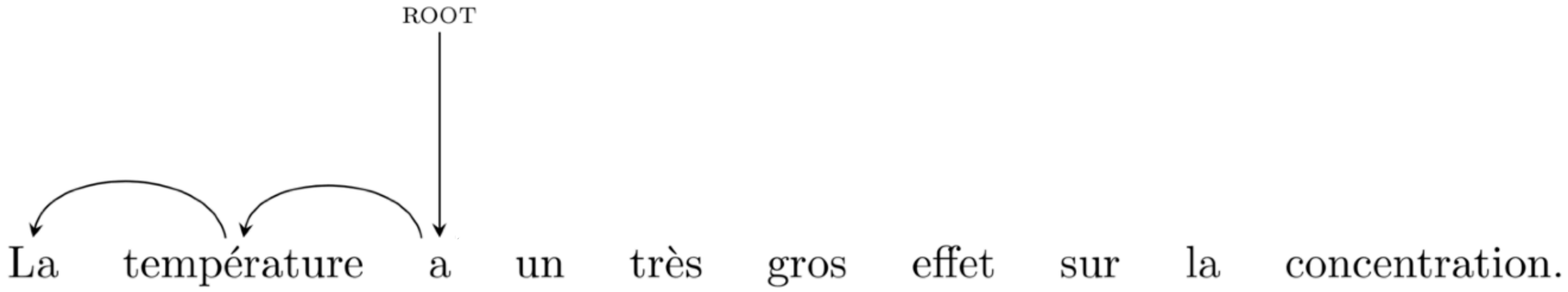
action: Right-Arc()



Arc-eager on an example

[ROOT, a, un]$_S$ [très, gros, effet, sur, la, concentration]$_Q$

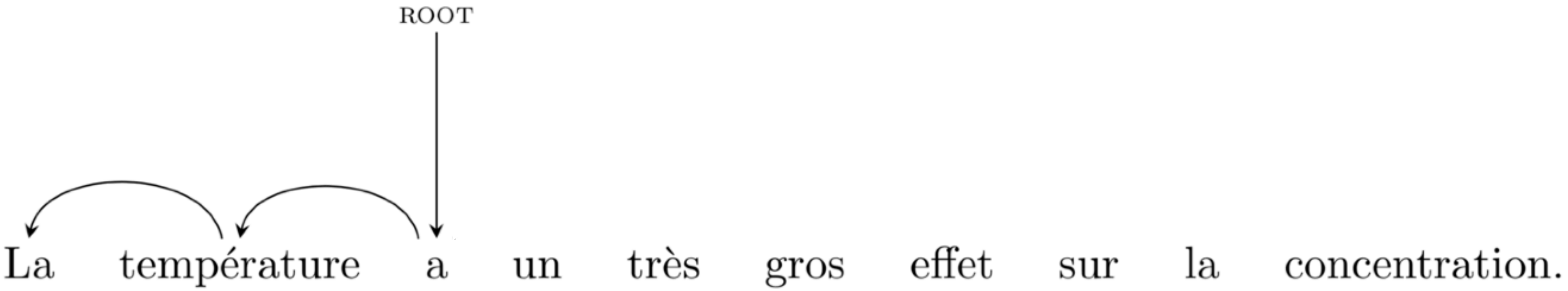
action: Shift



Arc-eager on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

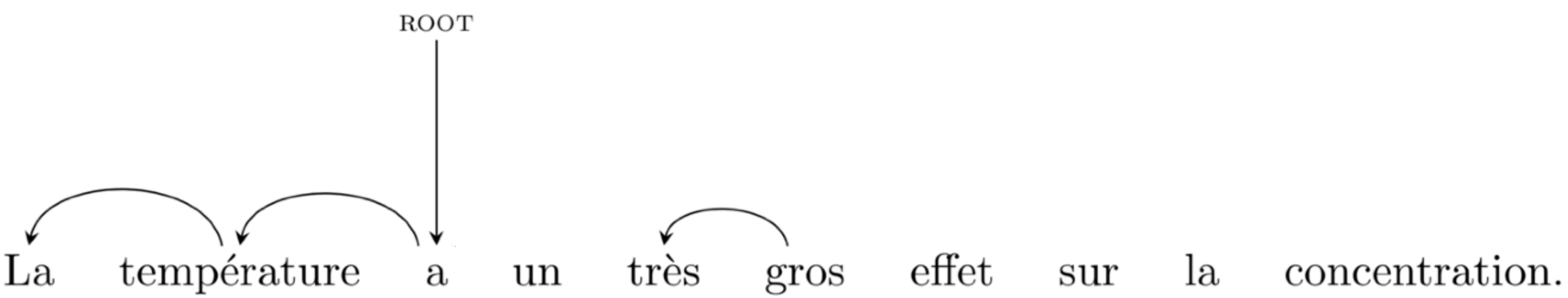
action: Shift



Arc-eager on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

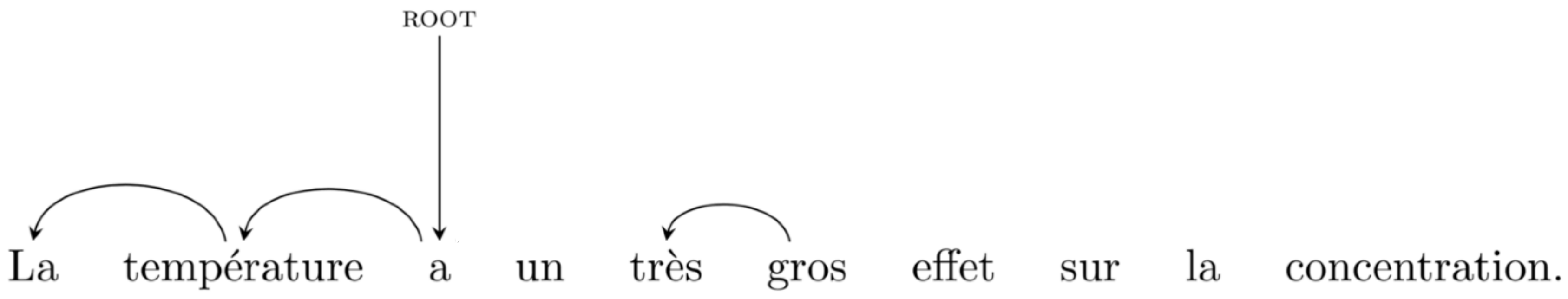
action: Left-Arc()



Arc-eager on an example

[ROOT, a, un, gros]$_S$ [effet, sur, la, concentration]$_Q$

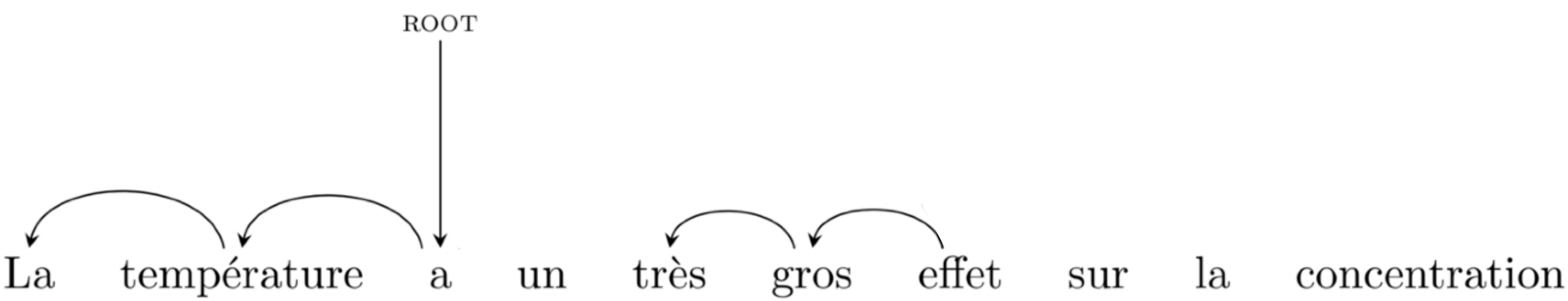
action: Shift



Arc-eager on an example

[ROOT, a, un, gros]$_S$ [effet, sur, la, concentration]$_Q$

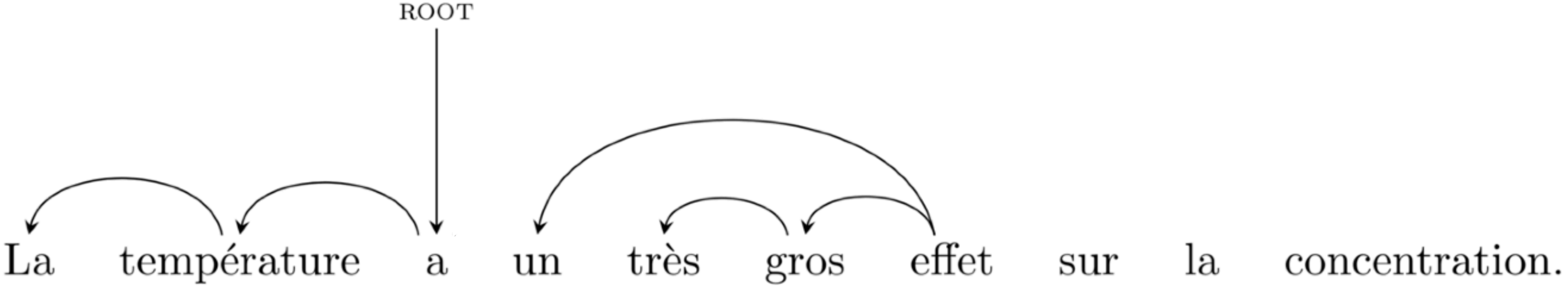
action: Left-Arc()



Arc-eager on an example

[ROOT, a, un]$_S$ [effet, sur, la, concentration]$_Q$

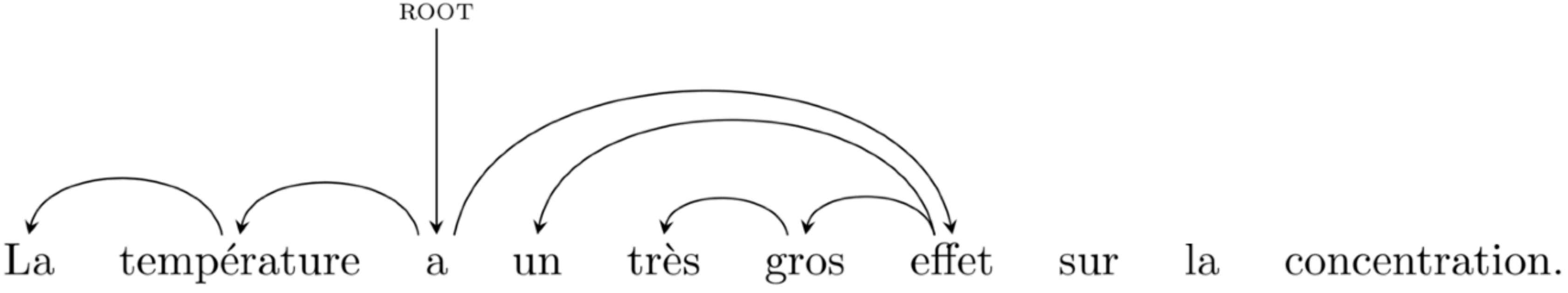
action: Left-Arc()



Arc-eager on an example

[ROOT, a, effet]$_S$ [sur, la, concentration]$_Q$

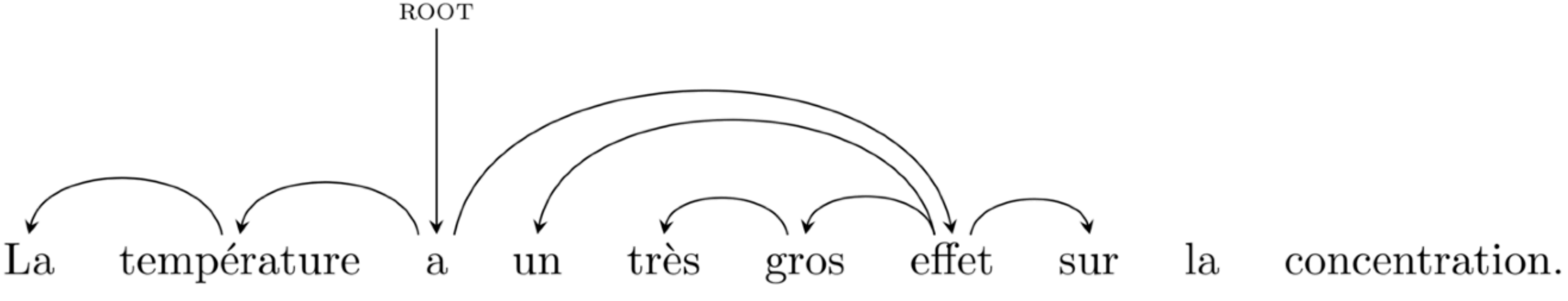
action: Right-Arc()



Arc-eager on an example

[ROOT, a, effet, sur]$_S$ [la, concentration]$_Q$

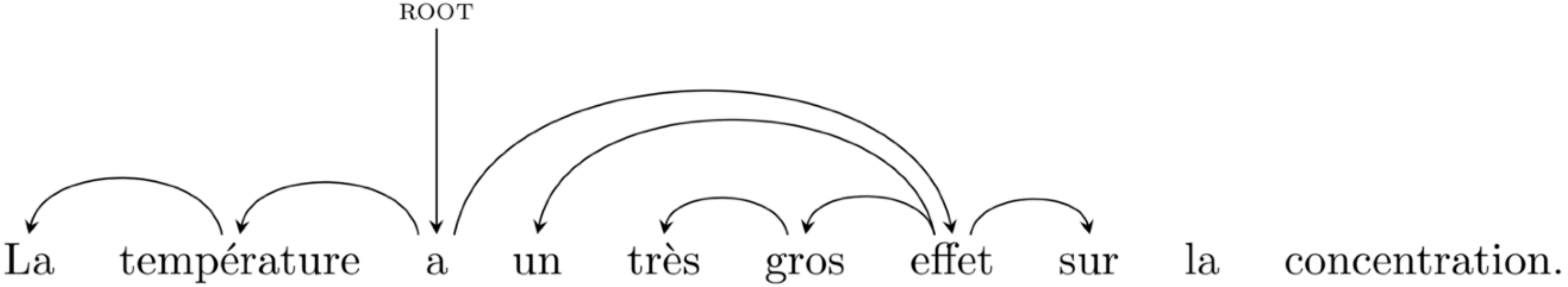
action: Right-Arc()



Arc-eager on an example

[ROOT, a, effet, sur, la]$_S$ [concentration]$_Q$

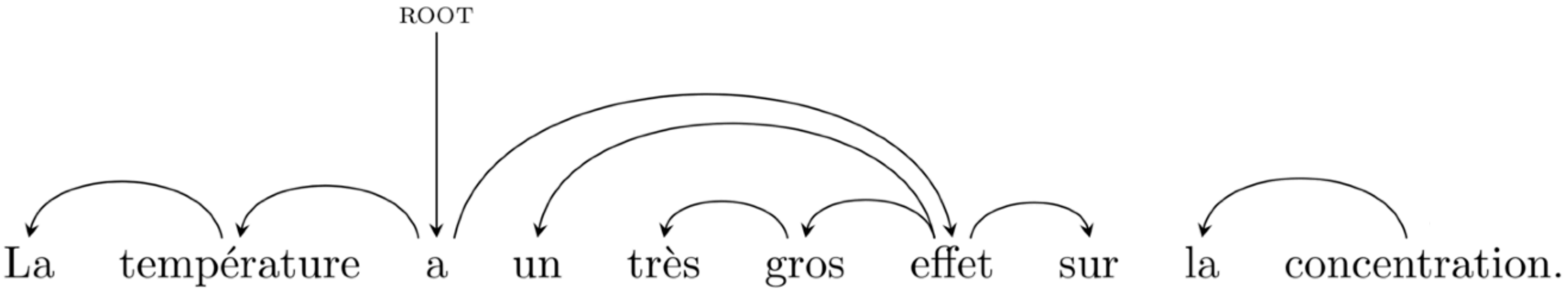
action: Shift()



Arc-eager on an example

[ROOT, a, effet, sur, la]$_S$ [concentration]$_Q$

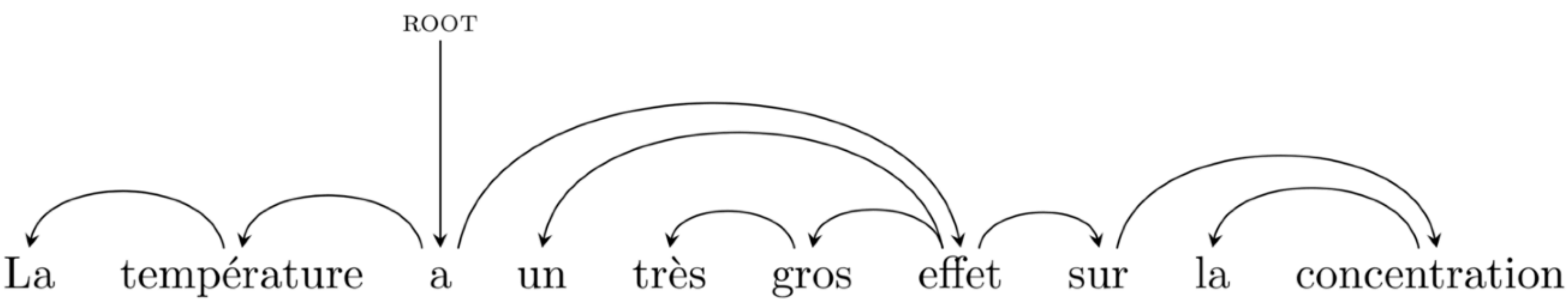
action: Left-Arc()



Arc-eager on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

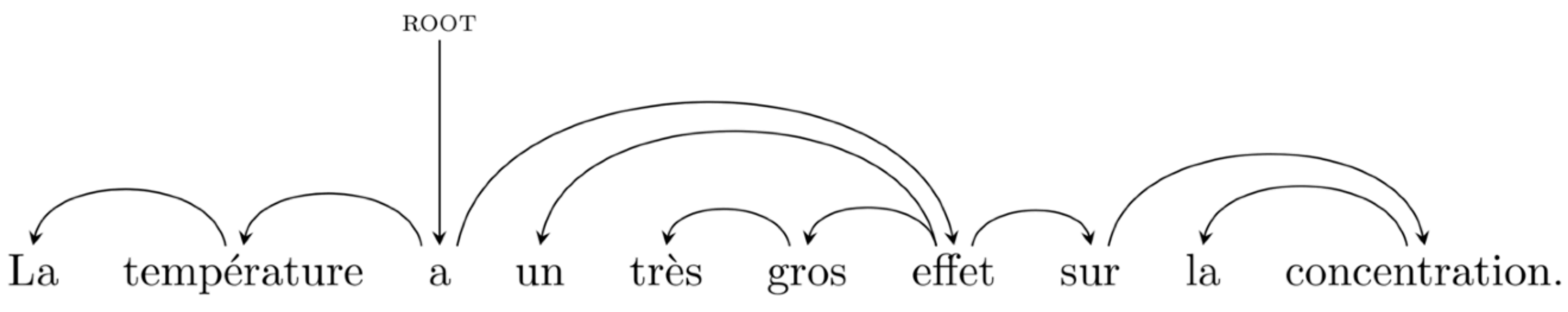
action: Right-Arc()



Arc-eager on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

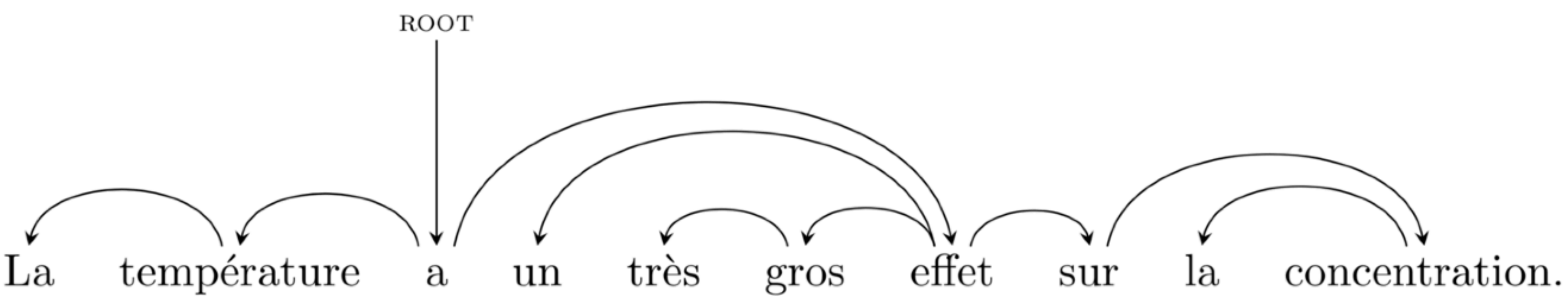
action: Reduce



Arc-eager on an example

[ROOT, a, effet, sur]$_S$ []$_Q$

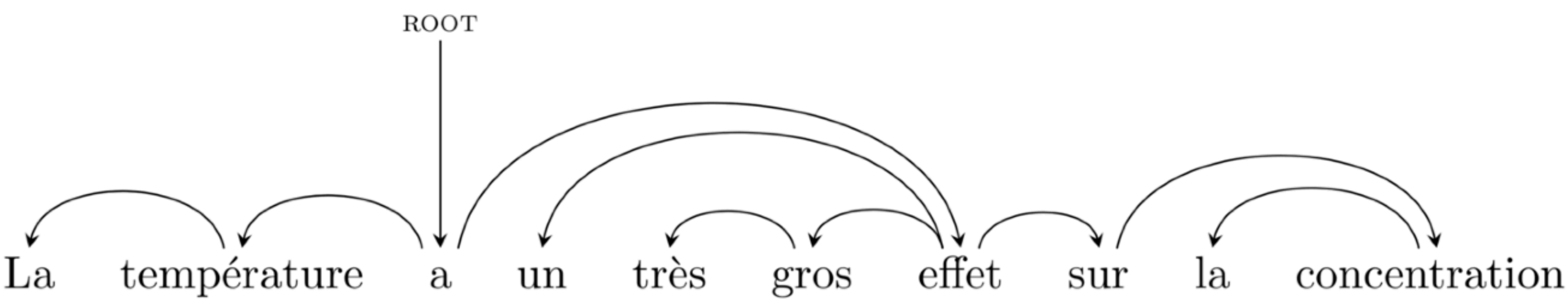
action: Reduce



Arc-eager on an example

[ROOT, a, effet]$_S$ []$_Q$

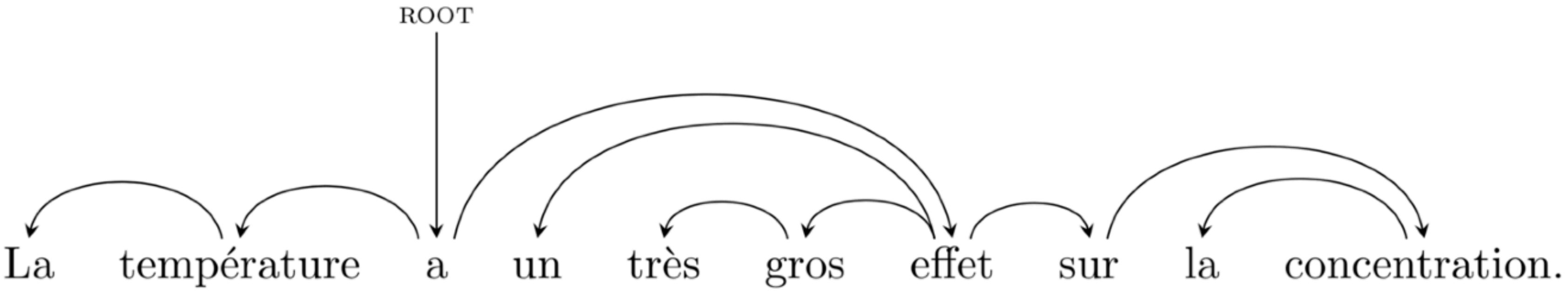
action: Reduce



Arc-eager on an example

[ROOT, a]$_S$ []$_Q$

action: Reduce



Drawbacks of the arc-eager algorithm

- The arc-eager system has a weaker soundness result than the arc-standard system
- It does not guarantee the output to be a dependency tree, only a sequence of (unconnected) trees.
- In the best case, this is a sequence of length 1, meaning that the tree is in fact a tree.
- In the worst case, this is a sequence of length n, meaning that each word is its own tree.
- The arc-eager parsers normally have a last step that attaches everything that remains in the stack to the root

Transition-based parsing with a neural classifier

The problem with manual features

- Feature combinations yield literally **millions of features for parsing**
- It’s very difficult to weight them all correctly or to chose the right **feature templates**
- Despite being many, they are still always incomplete
- Lexical features are extremely sparse:
  - the feature ‘word surface form’ can take any of **tens or hundreds of thousands categorical values**…
  - …each absolutely unique and not related to each other
- In the end, **feature extraction sometimes takes more time than parsing itself**

Example of a neural arc-standard algorithm

- Chen and Manning (2014)
  - The first neural parsing architecture that really works
- Replace the action selection module by a neural network

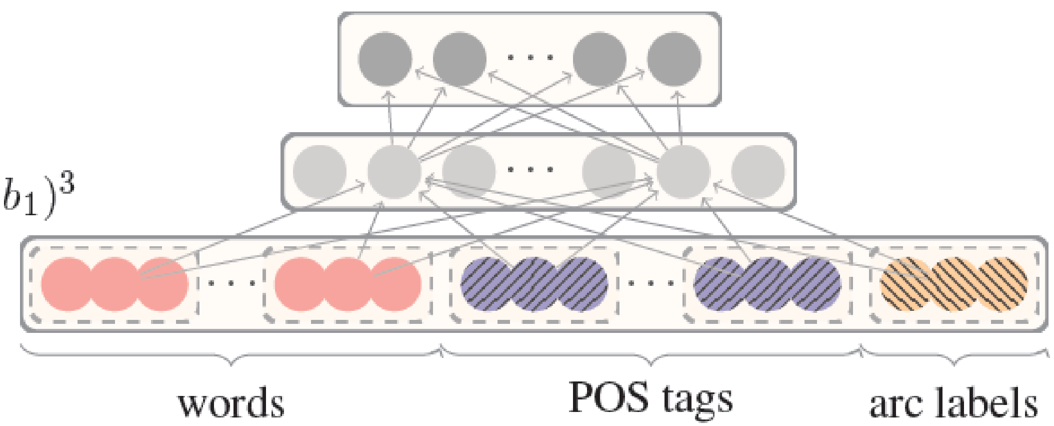
Softmax layer:

$$p = \mathtt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

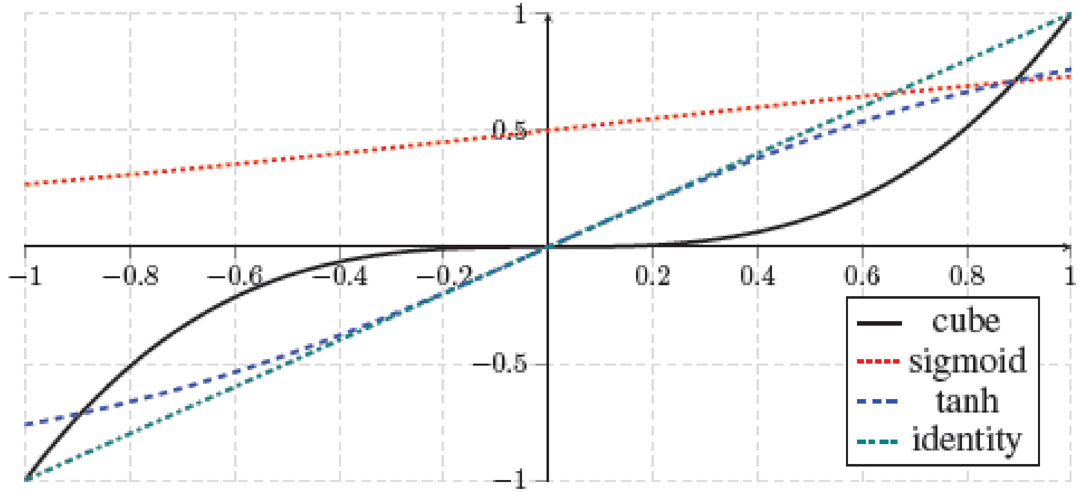
Input layer: $[x^w, x^t, x^l]$



Example of a neural arc-standard algorithm

- Chen and Manning (2014)
  - The first neural parsing architecture that really works
- Replace the action selection module by a neural network
- Cube activation function
  - It directly extracts feature combinations of up to three features

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$



Example of a neural arc-standard algorithm

- Chen and Manning (2014)
  - The first neural parsing architecture that really works
- Replace the action selection module by a neural network
- Cube activation function
- POS tags and arc labels are discrete sets
  - Normally represented as one-hot vectors
  - Just like words, there should be similarities
    - NN (singular noun) should be similar to NNP (plural noun)
  - Dense embedding layer for POS tags and arc labels capture relationships
- **Better results in accuracy and parsing speed** compared with previous parsers with statistical classifiers

Example of a neural arc-standard algorithm: Experimental Results

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	89.9	88.7	89.7	88.3	51
eager	90.3	89.2	89.9	88.6	63
Malt:sp	90.0	88.8	89.9	88.5	560
Malt:eager	90.1	88.9	90.1	88.7	535
MSTParser	92.1	90.8	**92.0**	90.5	12
Our parser	**92.2**	**91.0**	**92.0**	**90.7**	**1013**

Table 4: Accuracy and parsing speed on PTB + CoNLL dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	**92.0**	**89.7**	**91.8**	**89.6**	**654**

Table 5: Accuracy and parsing speed on PTB + Stanford dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	82.4	80.9	82.7	81.2	72
eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	**84.0**	82.1	83.0	81.2	6
Our parser	**84.0**	**82.4**	**83.9**	**82.4**	**936**

Table 6: Accuracy and parsing speed on CTB.

Example of a neural arc-standard algorithm: Model comparison

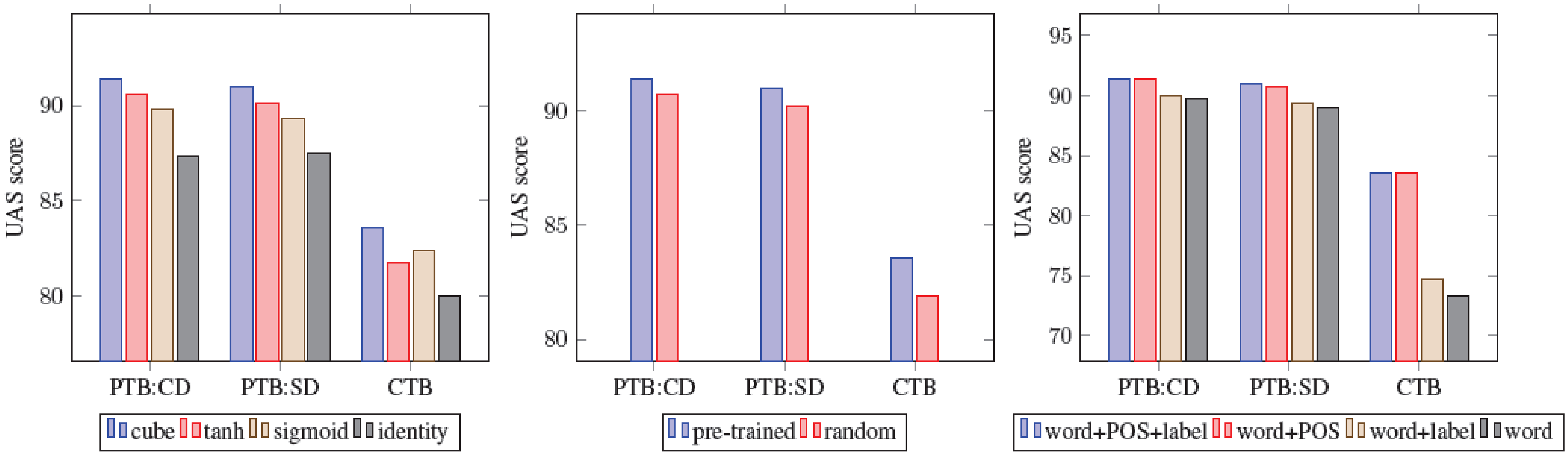


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

Deep learning for parsing

SyntaxNet

- In 2016, Google released SyntaxNet, a neural parser implemented in TensorFlow, and state-of-the-art models:
  - https://github.com/tensorflow/models/tree/master/research/syntaxnet
- Implements the system described in (Andor et al. 2016):
  - ‘globally normalized transition-based dependency parser’
  - Changes compared to (Chen and Manning 2014):
    - Beam search
    - **Global optimisation** using Conditional Random Fields (CRF)
      - all valid sequences of transition operators are scored.
    - 2 hidden layers of 1024 dimensions each.
- Combines the flexibility of transition-based algorithms and the modelling power of neural networks (even without recurrence)
- Parsey McParseface model: **92.79 LAS on English PTB**
  - **LAS 80.38 on UD v1.3 English Treebank**

ParseySaurus

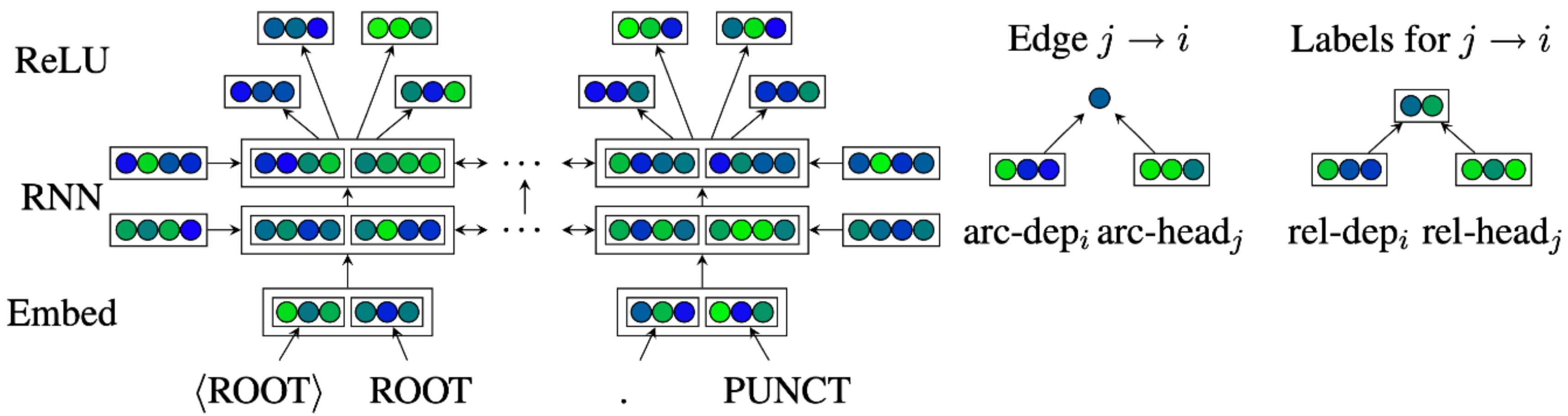
- Google then moved to using LSTMs in their **DRAGNN** framework
  - ‘Dynamic Recurrent Acyclic Graphical Neural Networks ’;
  - Described in (Alberti et al. 2017)
  - LSTM transition-based neural model
  - character-based input layer
- ParseySaurus model: **84.45 LAS on UD v1.3 English Treebank**

The CoNLL 2017 shared task

- **The task was to parse raw texts in different languages into dependency trees**
- Unlike the previous CoNLL 2007 shared task, **the input is raw text**:
  - no tokenisation
  - no sentence segmentation
  - no lemmas
  - no PoS tags
- **Consistent *Universal Dependencies* (UD) annotation used for all languages**
- Training and test data came from the UD 2.0 collection:
  - 64 treebanks in 45 languages.
- 4 ‘surprise’ languages with no training data: Buryat, Kurmanji Kurdish, North Saami and Upper Sorbian
- A major milestone in advancing data-driven dependency parsing
  - 33 participants
  - DRAGNN was one of the 2 baselines

The Dozat et al. (2017) parser

- The system described in (Dozat et al. 2017) is the winner of the shared task
  - average LAS 76.30, average UAS 81.30
- **Graph-based**: for each word, the parser looks for the most likely head, and then decides how to label the resulting dependency



The Dozat et al. (2017) parser

- The input to the model is a sequence of tokens and their part of speech tags
  - Word embeddings + character-based embeddings
- It is put through a 3-layer bidirectional LSTM network
- The output state of the final LSTM layer is then fed through four separate ReLU layers, producing four specialised vector representations for each word
  1. one for the word as a dependent seeking its head
  2. one for the word as a head seeking all its dependents
  3. another for the word as a dependent deciding on its label
  4. and a fourth for the word as head deciding on the labels of its dependents
- These vectors are then sequentially fed to two biaffine classifiers:
  - the first computes a score for each pair of tokens, with the highest score for a given token indicating that token's most probable head
  - the second computes a score for each label for a given token/head pair, with the highest score representing the most probable label for the arc from the head to the dependent

Beyond the Dozat et al. (2017) parser

- In the CoNLL 2017 shared task, Dozat and colleagues used word2vec (non-contextual) word embeddings
- They can be replaced with contextual embeddings (ELMo, BERT)
- But the contextual information provided by BERT makes the LSTM layers redundant
- The output of BERT can replace the architecture up to the LSTM layers (included)
  - This is the parsing architecture proposed by (Kondratyuk & Straka 2019)
  - It is the architecture we used to evaluate the parsing performance of our French BERT model CamemBERT (Martin et al. 2019, 2020)
    - We improve the state of the art of parsing for French

Time for a short break!