Transition-based Dependency Parsing

Miguel Ballesteros

Algorithms for NLP Course.
7-11

By using some materials of Joakim Nivre from Uppsala University

Transition-based Dependency Parsing

- Transition-based models for dependency parsing use a factorization defined in terms of a transition system, or abstract state machine.
- We will introduce some transition systems for dependency parsing.
- Different approaches to learning and decoding with these models: greedy classifier-based parsing and beam search, structured learning, and dynamic oracles.
- I will discuss different techniques for non-projective transition-based parsing.

Transition-based Dependency Parsing

1- Greedy transition-based parsing.

- Different algorithms.

2- Beam-search and structured learning.

3- Dynamic oracles.

4- Non-projective parsing
(algorithms and pseudo-projective parsing)

5- Joint Parsing and POS tagging.

6- Joint Parsing and other NLP tasks.

Transition Systems

- A transition system for dependency parsing is a quadruple $S = (C, T, c_s , C_t)$, where
  - C is a set of configurations.
  - T is a set of transitions, each of which is a (partial) function $t: C \rightarrow C$.
  - c_s is an initialization function, mapping a sentence x to its initial configuration $c_s(x)$.
  - $C_t \subseteq C$ is a set of terminal configurations.

Transition Systems

- A *configuration* for a sentence x is a triple c = (Σ, B, A), where:
  - Σ is a stack of nodes in $V_{x,}$ known as the **Stack.**
  - B is a list of nodes in V_x, known as the **Buffer.**
  - A is a set of dependency arcs in $V_x \times L \times V_x$ (for some set L of dependency labels).

Transition Systems

- A *transition sequence* for a sentence x in transition systems $S = (C, T, c_s , C_t)$ is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations.
  - $C_0 = c_s(x)$.
  - $c_m \in C_t$.
  - For every i $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.

Transition Systems

- The parse assigned to x by $C_{0,m}$ is the dependency graph $G_{cm} = (V_x , A_{cm})$,
  - A_{cm} is the set of dependency arcs in cm .
  - More generally, the dependency graph associated with any configuration c_i for x is $G_{ci} = (V_x, A_{ci})$.

Transition Systems

- **Oracle**: an oracle is a deterministic (or may be non-deterministic) algorithm that taking a gold tree associated to a sentence, it produces the set of actions the algorithm should follow in order to reach the gold tree.

Transition Systems

- We will consider several transition systems:
  - Nivre's Arc-Standard.
  - Nivre's Arc-eager.
  - Non-projectivity with Attardi's.
  - Non-projectivity with Arc-standard.
  - Joint part-of-speech tagging and dependency parsing.
  - Joint (any task) and dependency parsing.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, \text{label}, i)\})$ 1 (**Left-Arc** label) Permitted only if $i \neq 0$.
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, \text{label}, j)\})$ (**Right-Arc** label)

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0],\ [1,\ .\ .\ .\ ,\ n],\ \varnothing)$

We start with a stack with the root symbol, and a buffer full of words.
The “arcs”-structure is empty.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0],\ [1,\ .\ .\ .\ ,\ n],\ \varnothing)$

We start with a stack with the root symbol, and a buffer full of words

.

A Spanish guy called Miguel :-) (with Joakim Nivre) proved
that it is better to put the root at the end of the buffer as a starting position
(Ballesteros & Nivre 2013, Computational Linguistics) like this:

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = (\varnothing,\ [1,\ .\ .\ .\ ,\ n,\ 0],\ \varnothing)$

It is also possible to not have any root at all. In the last step of the parsing process, we will basically attach to the root everything that remains in the stack (which is equivalent to have the root at the end) – and the example is going to be like this.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

The terminal configuration is when we have an empty buffer and the stack contains only the root. The set of arcs A is not empty anymore.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots\ , x_n\ ) = ([0], [1, \ldots\ , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)

The SHIFT action takes the first incoming word from the buffer and push it onto the stack.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, \text{label}, i)\})$ 1 (**Left-Arc** label)
  Permitted only if $i \neq 0$.

The left-arc action takes the two top words at the top of the stack and creates a left-arc between them. It removes the dependent (i) from the stack.

This action is permitted if the word i (the dependent) is not the root (id=0)

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, \text{label}, i)\})$ 1 (**Left-Arc** label) Permitted only if $i \neq 0$.
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, \text{label}, j)\})$ (**Right-Arc** label)

The right-arc action takes the two top words of the stack and creates a right-arc between them. It removes the dependent (j) from the stack.

Nivre's Arc-Standard

Initialization:

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \varnothing)$

Terminal Conf:

- $C_t = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, \text{label}, i)\})$ 1 (**Left-Arc** label) Permitted only if $i \neq 0$.
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, \text{label}, j)\})$ (**Right-Arc** label)

Nivre's Arc-Standard Oracle

$c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0],\ [1, \ldots ,\ n],\ \varnothing)$

while (β **is not** empty and !(**σ** has only the root))

If (σ[0] is head of σ[1] and all childs of σ[1] are attached to it and σ[1] is not the root) **then** → **Left-Arc**

else if (σ[1] is head of σ[0] and all childs of σ[0] are attached to it and σ[0] is not the root) **then** → **Right-Arc**

else → **SHIFT**

Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**

They told him a story

Nivre's arc-standard

Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**

Nivre's arc-standard

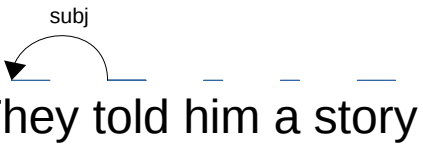
Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**

Nivre's arc-standard

Nivre's Arc-Standard

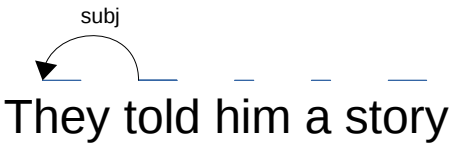
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**



Nivre's arc-standard

Nivre's Arc-Standard

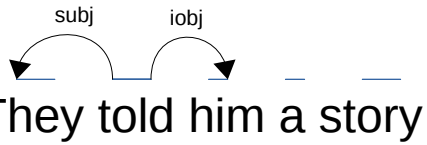
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**



Nivre's arc-standard

Nivre's Arc-Standard

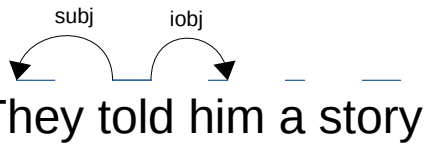
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**



Nivre's arc-standard

Nivre's Arc-Standard

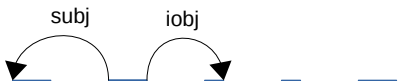
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**



Nivre's arc-standard

Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
SHIFT	**[told, a, story]**	**[ ]**

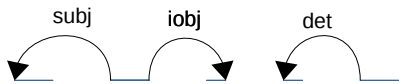


They told him a story

Nivre's arc-standard

Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
SHIFT	**[told, a, story]**	**[ ]**
LA (det)	**[told, story]**	**[ ]**



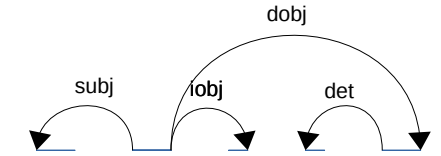
They told him a story

Nivre's arc-standard

Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
SHIFT	**[told, a, story]**	**[ ]**
LA (det)	**[told, story]**	**[ ]**
RA (dobj)	**[told]**	**[ ]**

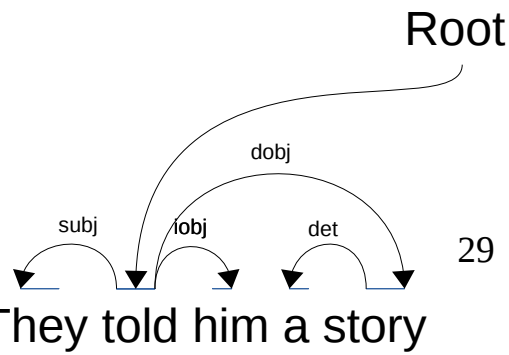
Nivre's arc-standard



Nivre's Arc-Standard

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
SHIFT	**[They, told]**	**[him a story]**
LA (subj)	**[told]**	**[him a story]**
SHIFT	**[told, him]**	**[a story]**
RA (iobj)	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
SHIFT	**[told, a, story]**	**[ ]**
LA (det)	**[told, story]**	**[ ]**
RA (dobj)	**[told]**	**[ ]**

Note that we do not have a dummy root node during parsing. The last token in the stack will be attached to the root.



Nivre's Arc-Standard

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

Nivre's Arc-Standard

- 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.

Nivre's Arc-Eager

- The arc-eager system always adds an arc at the earliest possible opportunity and it therefore builds **parts** of the tree **top-down** instead of **bottom-up.**

Nivre's Arc-Eager

- Initialization:
  $c_s\ (x = x_1\ , \ldots ,\ x_n\ ) = ([0], [1, \ldots , n], \emptyset)$
- Terminal:
  $C_t = \{c \in C | c = (\Sigma, [\ ], A)\}$
- Transitions:
  - $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
  - $([\sigma|i], [j|\beta], A) \Rightarrow (\sigma, [j|\beta], A \cup \{(j, label, i)\})$ (**Left-Arc** label)

    Permitted only if $i \neq 0$ and there are **no** k,label' such that $(k, label', i) \in A$
  - $([\sigma|i], [j|\beta], A) \Rightarrow ([\sigma|i|j], \beta, A \cup \{(i, label, j)\})$ (**Right-Arc** label)
  - $([\sigma|i], \beta, A) \Rightarrow (\sigma, \beta, A)$ 2 (**Reduce**)

    Permitted only if there are k,label' such that $(k, label', i) \in A$

Nivre's Arc-Eager

- Initialization:
  c s (x = x 1 , . . . , x n) = ([0], [1, . . . , n], ∅)

We start with a stack with the root symbol, and a buffer full of words.
Again, we can choose to not have any root at all, or to have at the end of the buffer.
We will do the examples without ROOT.

In arc-eager it is actually much better to put the root at the end. The reason is that the algorithm is too eager and it makes early attachments to the root node. It is better to postpone them. If we put the root at the end, we are doing that.

Nivre's Arc-Eager

- Initialization:
  $c\ s\ (x = x\ 1\ , \ldots , x\ n\ ) = ([0], [1, \ldots , n], \emptyset)$
- Terminal:
  $C\ t = \{c \in C | c = (\Sigma, [\ ], A)\}$

The terminal configuration is when we have an empty buffer and the stack contains input symbols. The set of arcs A is not empty anymore.

Nivre's Arc-Eager

- Initialization:
  $c\ s\ (x = x\ 1\ , \ldots , x\ n\ ) = ([0], [1, \ldots , n], \emptyset)$
- Terminal:
  $C\ t = \{c \in C | c = (\Sigma, [\ ], A)\}$
- Transitions:
  - $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (Shift)

The SHIFT action takes the first incoming word from the buffer and push it onto the stack.

Nivre's Arc-Eager

- Initialization:
  $c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0], [1, .\ .\ .\ , n], \varnothing)$
- Terminal:
  $C_t = \{c \in C | c = (\Sigma, [\ ], A)\}$
- Transitions:
  - $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
  - $([\sigma|i], [j|\beta], A) \Rightarrow (\sigma, [j|\beta], A \cup \{(j, label, i)\})$ (**Left-Arc** label)

    Permitted only if $i \neq 0$ and there are **no** k,label' such that $(k, label', i) \in A$

The Left-arc action takes the first incoming word from the buffer and the word at the top of the stack and create left-arc with head j and dependent i. It later erased the word i from the stack (reduce).

This is permitted if the dependent is not the root and there is no incoming arcs to the dependent.

Nivre's Arc-Eager

- Initialization:
  $c_s\ (x = x_1\ , \ldots\ , x_n\ ) = ([0], [1, \ldots\ , n], \emptyset)$
- Terminal:
  $C_t = \{c \in C | c = (\Sigma, [\ ], A)\}$
- Transitions:
  - $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
  - $([\sigma|i], [j|\beta], A) \Rightarrow (\sigma, [j|\beta], A \cup \{(j, label, i)\})$ (**Left-Arc** label)

    Permitted only if $i \neq 0$ and there are **no** k,label' such that $(k, label', i) \in A$
  - $([\sigma|i], [j|\beta], A) \Rightarrow ([\sigma|i|j], \beta, A \cup \{(i, label, j)\})$ (**Right-Arc** label)

The Right-arc action takes the top of the stack and the first word of the buffer, and it Creates and arc with head=top of the stack and dependent=first word of the buffer. It later **shifts** the first incoming word from the buffer and push it onto the stack.

Nivre's Arc-Eager

- Initialization:
  $c_s\ (x = x_1\ , \ldots\ , x_n\ ) = ([0], [1, \ldots\ , n], \emptyset)$
- Terminal:
  $C_t = \{c \in C | c = (\Sigma, [\ ], A)\}$
- Transitions:
  - $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
  - $([\sigma|i], [j|\beta], A) \Rightarrow (\sigma, [j|\beta], A \cup \{(j, label, i)\})$ (**Left-Arc** label)

    Permitted only if $i \neq 0$ and there are **no** k,label' such that $(k, label', i) \in A$
  - $([\sigma|i], [j|\beta], A) \Rightarrow ([\sigma|i|j], \beta, A \cup \{(i, label, j)\})$ (**Right-Arc** label)
  - $([\sigma|i], \beta, A) \Rightarrow (\sigma, \beta, A)$ 2 (**Reduce**)

    Permitted only if there are k,label' such that $(k, label', i) \in A$

The Reduce action removes the top of the stack (only) if it already has a head, and if we are done with it.

Nivre's Arc-Eager Oracle

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0],\ [1,\ .\ .\ .\ ,\ n],\ \varnothing)$

while (β **is not** empty)

If (β[0] is head of σ[0]) **then** → **Left-Arc**

else If (σ[0] is head of β[0]) **then** → **Right-Arc**

else If (all childs of σ[0] are attached to it and σ[0] has a head) **then** → **Reduce**

else → **SHIFT**

Nivre's Arc-Eager Oracle

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0],\ [1,\ .\ .\ .\ ,\ n],\ \varnothing)$

while (β **is not** empty)

If (β[0] is head of σ[0]) **then** → **Left-Arc**

else If (σ[0] is head of β[0]) **then** → **Right-Arc**

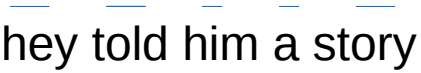
else If (all childs of σ[0] are attached to it and σ[0] has a head) **then** → **Reduce**

else → **SHIFT**

Note that, in contrast with arc-std, we do not care about the state of the stack

Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**



Nivre's arc-standard

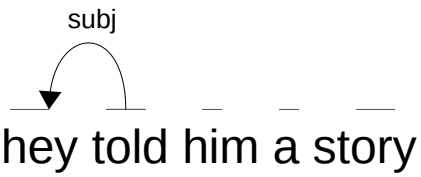
Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**

Nivre's arc-standard

Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**



Nivre's arc-standard

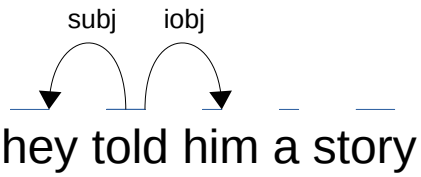
Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**

Nivre's arc-standard

Nivre's Arc-Eager

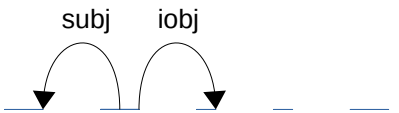
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**



Nivre's arc-standard

Nivre's Arc-Eager

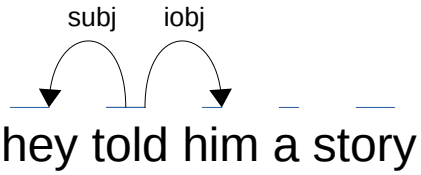
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**
REDUCE	**[told]**	**[a story]**



Nivre's arc-standard

Nivre's Arc-Eager

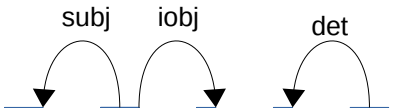
Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**
REDUCE	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**



Nivre's arc-standard

Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**
REDUCE	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
L-A(det)	**[told]**	**[story]**



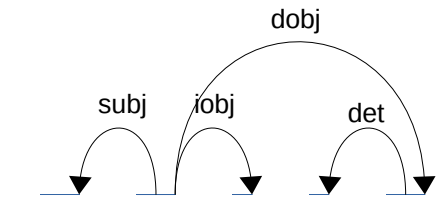
They told him a story

Nivre's arc-standard

Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**
REDUCE	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
L-A(det)	**[told]**	**[story]**
R-A(dobj)	**[told, story]**	**[ ]**

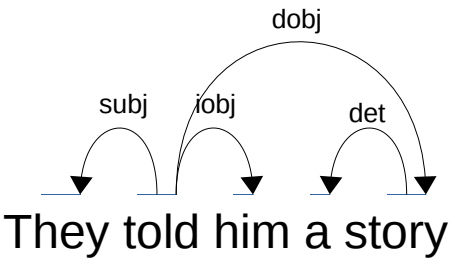
Nivre's arc-standard



Nivre's Arc-Eager

Action	**Stack**	**Buffer**
	[ ]	**[They told him a story]**
SHIFT	**[They]**	**[told him a story]**
L-A (subj)	**[ ]**	**[told him a story]**
SHIFT	**[told]**	**[him a story]**
R-A (iobj)	**[told, him]**	**[a story]**
REDUCE	**[told]**	**[a story]**
SHIFT	**[told, a]**	**[story]**
L-A(det)	**[told]**	**[story]**
R-A(dobj)	**[told, story]**	**[ ]**
REDUCE	**[told]**	**[ ]**

Nivre's arc-standard



Nivre's Arc-Eager

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

Nivre's Arc-Eager

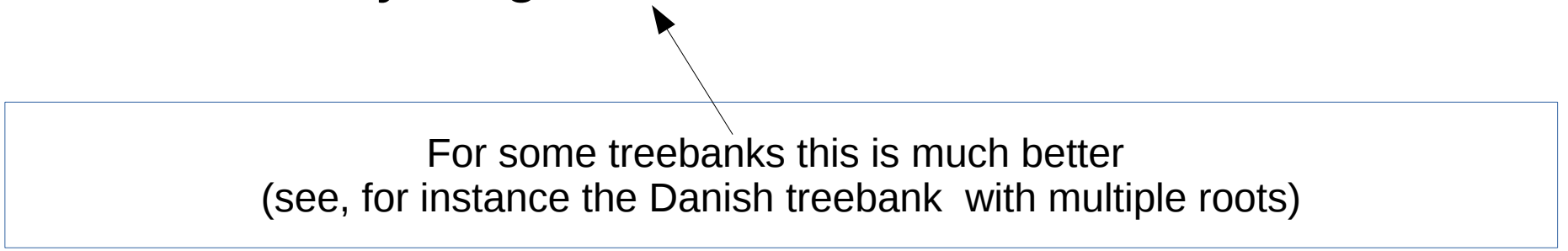
- The arc-eager system has a weaker soundness result than the arc-standard system
  - It does not guarantee the output to be a dependency tree, only a sequence of (unconnected) trees (a forest).
  - In the best case, this is a sequence of length 1, meaning that the tree is in fact a tree.
  - In the worst case, this is a sequence of length n, meaning that each words is its own tree.

  The arc-eager parsers normally have a last step that attaches everything that remains in the stack to the root.

Nivre's Arc-Eager

- 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 (a forest).
  - 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 words is its own tree.

The arc-eager parsers normally have a last step that attaches everything that remains in the stack to the root.



Greedy Transition-Based Parsing

- In a transition-based model, dependency trees are represented by the transition sequences that derive them → **Oracle**.
- The transition system itself is nondeterministic (there could be several transitions that provide a solution to the same tree) and does not impose any preference among possible transition sequences for a given sentence.
- In order to turn this into a parser, we have to score transition sequences and a method for finding the highest-scoring sequence under our current model.

Greedy Transition-Based Parsing

- In the greedy perspective, we assume that the problem of scoring transition sequences can be reduced to the simple problem of scoring single transitions.
- This is also called deterministic transition-based parsing.

- Remember the local discriminative models for PCFGs (a week ago).

$$P(y|x) = \prod_{i=1}^{m} P(d_i|\Phi(d_1, \ldots, d_{i-1}, x))$$

Greedy Transition-Based Parsing

Parse (x = (w0 , w1 , . . . , wn)

c ← cs (x)

while c is not in Ct

t* ← argmax_t Score(c, t)

c ← t*(c)

return Gc

Greedy Transition-Based Parsing

Parse (x = (w0 , w1 , . . . , wn)

c ← cs (x)

while c is not in Ct

t* ← argmax_t Score(c, t)

c ← t*(c)

return Gc

After initializing the parser to the initial configuration, the algorithm consists of a single loop that just repeatedly applies the highest-scoring transition out of the current configuration until a terminal configuration is reached.

Greedy Transition-Based Parsing

- The worst-case complexity of parsing is linear in the length of the sentence **O(n)**.
  - Given that the computation of the highest-scoring transition to the current configuration can be performed in constant time.

Greedy Transition-Based Parsing

- A wide range of different models have been used to score transitions, but the most common approach is a linear model:

$$\text{Score}(c,t) = \sum_{k=1} \mathbf{f}_k(c,t) \cdot \mathbf{w}_k$$

Greedy Transition-Based Parsing

- Typical feature templates of greedy transition-based parsers.

Unigrams	**Bigrams**
Σ_0.pos	Σ_0.pos, B_0.pos
Σ_1.pos	Σ_0.pos, Σ_0.lab
B_0.pos	B_0.pos, LDEP(B_0).lab
B_1.pos	
B_2.pos	
B_3.pos	**Trigrams**
Σ_0.lab	Σ_1.pos, Σ_0.pos, B_0.pos
LDEP(Σ_0).lab	Σ_0.pos, B_0.pos, B_1.pos
RDEP(Σ_0).lab	B_0.pos, B_1.pos, B_2.pos
LDEP(B_0).lab	B_1.pos, B_2.pos, B_3.pos
Σ_0.word	Σ_0.pos, LDEP(Σ_0).lab, RDEP(Σ_0).lab
B_0.word	
B_1.word	
HD(Σ_0).word	

Greedy Transition-Based Parsing

- Typical feature templates of greedy transition-based parsing.

Unigrams	Bigrams
Σ_0.pos	Σ_0.pos, B_0.pos
Σ_1.pos	Σ_0.pos, Σ_0.lab
B_0.pos	B_0.pos, LDEP(B_0).lab
B_1.pos	
B_2.pos	
B_3.pos	**Trigrams**
Σ_0.lab	Σ_1.pos, Σ_0.pos, B_0.pos
LDEP(Σ_0).lab	Σ_0.pos, B_0.pos, B_1.pos
RDEP(Σ_0).lab	B_0.pos, B_1.pos, B_2.pos
LDEP(B_0).lab	B_1.pos, B_2.pos, B_3.pos
Σ_0.word	Σ_0.pos, LDEP(Σ_0).lab, RDEP(Σ_0).lab
B_0.word	

Note that the features are described to words that are either in the buffer or the stack.

Greedy Transition-Based Parsing

- Typical feature templates of greedy transition-based parsing.

Unigrams	Bigrams
Σ_0.pos	Σ_0.pos, B_0.pos
Σ_1.pos	Σ_0.pos, Σ_0.lab
B_0.pos	B_0.pos, LDEP(B_0).lab
B_1.pos	
B_2.pos	
B_3.pos	**Trigrams**
Σ_0.lab	Σ_1.pos, Σ_0.pos, B_0.pos
LDEP(Σ_0).lab	Σ_0.pos, B_0.pos, B_1.pos
RDEP(Σ_0).lab	B_0.pos, B_1.pos, B_2.pos
LDEP(B_0).lab	B_1.pos, B_2.pos, B_3.pos
Σ_0.word	Σ_0.pos, LDEP(Σ_0).lab, RDEP(Σ_0).lab
B_0.word	

Note that you can include (and you should) history-based features.

Greedy Transition-Based Parsing

- Greedy transition-based parsing was first introduced by Yamada&Matsumoto (2003) and Nivre (2003).
- They were very competitive in the CoNLL Shared Tasks of parsing in 2006 and 2007 (especially MaltParser by Nivre)
- Not anymore for 8 years...
- until 2015... in which they again produce state-of-the-art accuracy :-)
  - Chen & Manning. EMNLP 2014.
  - Dyer et al. ACL 2015.
  - Weiss et al. ACL 2015.
  - Ballesteros et al. EMNLP 2015.

Error Propagation

- Greedy transition-based dependency parsing has two problems.
  - Lack of backtracking.
  - Error propagation.

Error Propagation

- Greedy transition-based dependency parsing has two problems.
  - Lack of backtracking.
  - Error propagation.
- Here, there are three options:

  1. You come up with a better classifier that makes few mistakes (like LSTMs) :-)

  2. You do beam-search and structured learning.

  3. You implement dynamic oracles
  (we will later see what this is).

Beam Search and Structured Learning

- Beam search is a heuristic search algorithm that explores the q most promising hypotheses at each step.
- q is the beam size.
- q=1 is **greedy search**.
- With higher q we can explore a larger part of the search space.

Beam Search and Structured Learning

- Beam-search requires that we switch to a model that scores complete transition sequences:

$$\text{Score}(c_{0,m}, x) = \sum_{i=0}^{m} \text{Score}(c_i, t_i)$$

$$[\text{where } t_i(c_i) = c_{i+1}]$$

Beam Search and Structured Learning

- The overall algorithm is very similar.
- We initialize the beam to a set containing the single start configuration with score 0.0
- Then, we apply every possible transition t to the single start state, and we restrict the number of possibilities to the size of the beam (q highest-scoring ones).
- We do this for every new state we have created.
- We return the highest scoring path.

Beam Search and Structured Learning

```
PARSE(x = (w_0, w_1, ..., w_n))
1  BEAM ← {⟨c_s(x), 0.0⟩}
2  while ∃⟨c, s⟩ ∈ BEAM : c ∉ C_t
3      NEWBEAM ← ∅
4      for every (c, s) ∈ BEAM
5          for every t ∈ T
6              NEWBEAM ← NEWBEAM ∪ {⟨t(c), s+SCORE(c, t)⟩}
7      BEAM → QBEST(NEWBEAM)
8  return ← 1BEST(BEAM)
```

Beam Search and Structured Learning

- The accuracy of these beam parsers can be improved with early update (Collins & Roark 2004).
  - During training, parsing is interrupted for a giving path as soon as the gold standard transition falls out of the beam, and the weights are only updated up to this point.
  - “This is likely to lead to less noisy input to the parameter estimation algorithm; and it will also improve efficiency” (Collins & Roark 2004).

Beam Search and Structured Learning

- This technique is implemented in many parsers:
  - Zpar by Yue Zhang and Steve Clark.
  - Mate tools by Bernd Bohnet.
  - Google's Weiss et al. ACL 2015.

These are clear examples of beam-search transition-based parsers with state-of-the-art or close to state-of-the-art accuracy.

Dynamic Oracles

- The basic principle is that for some transition systems there is more than one path that could lead to the gold tree.
- In greedy parsing, we are only considering one of them, which is the one determined by the deterministic oracle.

- What if we care about all of them? (or at least a subset?)
  - Yoav Goldberg & Nivre.
  - Carlos Gómez-Rodriguez & Nivre
  - Ballesteros, Goldberg, Dyer and Smith 2015 (to appear. CL)

Dynamic Oracles

- **During training**

  Sometimes we flip a coin and we let the classifier to decide (and it might make mistakes or lead to a different gold path).

  - If gold-tree is reachable: allow actions leading to gold-tree.
  - If gold-tree is not reachable: allow actions leading to the best reachable tree.

  Best reachable tree: tree closer to the gold tree.

Dynamic Oracles

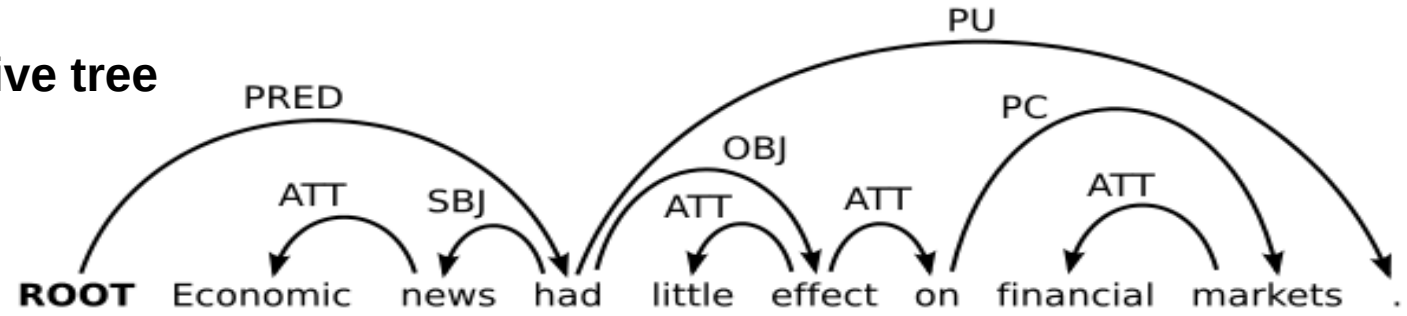
- In Decoding:

  We learn to parse in "difficult" configurations from which all (or many) gold arcs are still reachable.

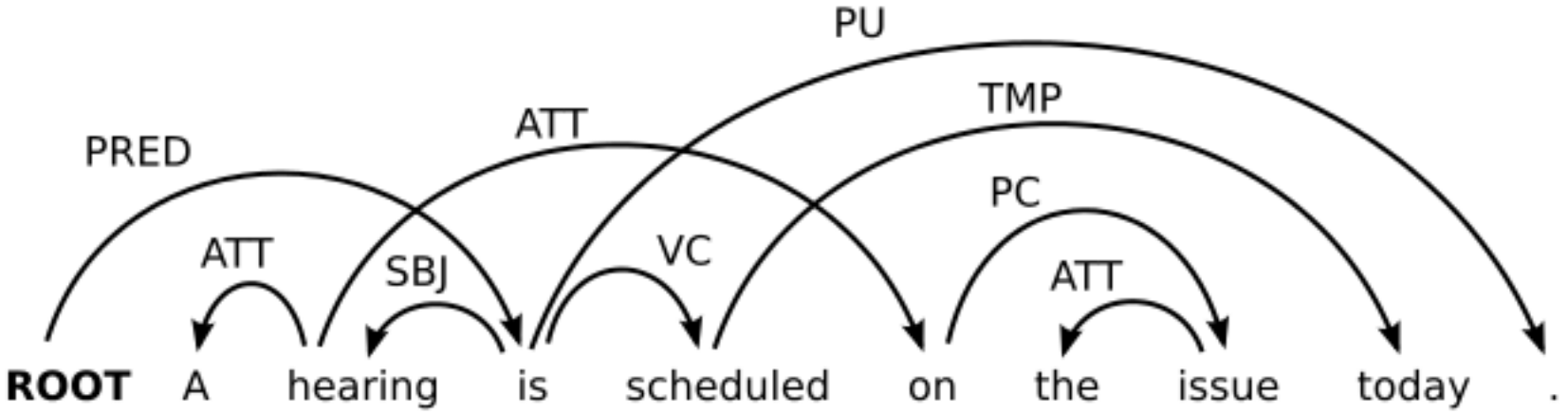
  But you also learn the gold path to the gold tree.

Remembering Projectivity

Projective tree



Non-projective tree



Parsing Non-Projective Trees

- With the algorithms studied so far, there is no way of parsing non-projective trees.
- Neither with arc-eager nor with arc-standard.
- We are going to study a couple of algorithms that are modifications to arc-standard that can output non-projective trees.
  - Attardi's algorithm.
  - Arc-standard with the SWAP transition.

Attardi's Algorithm

- It is an extension of arc-standard.

- It cannot handle arbitrary non-projective trees.
- But, it is sufficient to handle nearly all non-projective dependencies found in natural language data while maintaining the linear bound on transition sequence length.

Attardi's Algorithm

- Initialization: c s (x = x 1 , . . . , x n) = ([0], [1, . . . , n], ∅)
- Terminal: C t = {c ∈ C|c = ([0], [ ], A)}
- Transitions:

  (σ, [i|β], A) ⇒ ([σ|i], β, A) (Shift)

  ([σ|i|j], β, A) ⇒ ([σ|j], β, A∪{(j, l, i)}) 1 (Left-Arc l)

  ([σ|i|j], β, A) ⇒ ([σ|i], β, A∪{(i, l, j)}) (Right-Arc l)

  ([σ|i|k|j], β, A) ⇒ ([σ|k|j], β, A∪{(j, l, i)}) 1 (Left-Arc2 l)

  ([σ|i|k|j], β, A) ⇒ ([σ|i|k], β, A∪{(i, l, j)}) (Right-Arc2 l)

  ([σ|i|k_1|k_2 |j], β, A) ⇒ ([σ|k_1|k_2|j], β, A∪{(j, l, i)}) (Left-Arc3 l)

  ([σ|i|k_1|k_2|j], β, A) ⇒ ([σ|i|k_1|k_2], β, A∪{(i, l, j)}) (Right-Arc3 l)

Attardi's Algorithm

- Attardi's algorithm would require an infinite number of transitions to get full coverage.

Nivre's Arc-Standard with SWAP

Initialization:

$c\ s\ (x = x\ 1\ ,\ .\ .\ .\ ,\ x\ n\ ) = ([0], [1, .\ .\ .\ , n], \emptyset)$

Terminal:

$Ct = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

$(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, l, i)\})$ 1 (**Left-Arc** l)

Permitted only if $i \neq 0$.

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, l, j)\})$ (**Right-Arc** l)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], [i|\beta], A)$ 2 (**Swap**)

Permitted only if $i \neq 0$ and $i < j$.

Nivre's Arc-Standard Oracle

$c_s\ (x = x_1\ , \ldots\ , x_n\ ) = ([0], [1, \ldots, n], \varnothing)$

while (β **is not** empty and !(**σ** has only the root))

If (σ[0] is head of σ[1] and all childs of σ[1] are attached to it and σ[1] is not the root) **then** → **Left-Arc**

else if (σ[1] is head of σ[0] and all childs of σ[0] are attached to it and σ[0] is not the root) **then** → **Right-Arc**

else if (σ[1] > σ[0]) **then** → **SWAP**

else → **SHIFT**

This refers to the inorder traversal linear order.

Nivre's Arc-Standard with SWAP

Initialization:

$c\ s\ (x = x\ 1\ ,\ .\ .\ .\ ,\ x\ n\ ) = ([0], [1, .\ .\ .\ , n], \emptyset)$

- Terminal:

$Ct = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

- $(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift**)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, l, i)\})$ 1 (**Left-Arc** l)

  Permitted only if $i \neq 0$.
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, l, j)\})$ (**Right-Arc** l)
- $([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], [i|\beta], A)$ 2 (**Swap**)

  Permitted only if $i \neq 0$ and $i < j$.

This refers to the inorder traversal linear order.

Nivre's Arc-Standard Oracle

$c_s\ (x = x_1\ ,\ .\ .\ .\ ,\ x_n\ ) = ([0],\ [1,\ .\ .\ .\ ,\ n],\ \varnothing)$

while (β **is not** empty and !(**σ** has only the root))

If (σ[0] is head of σ[1] and all childs of σ[1] are attached to it and σ[1] is not the root) **then** → **Left-Arc**

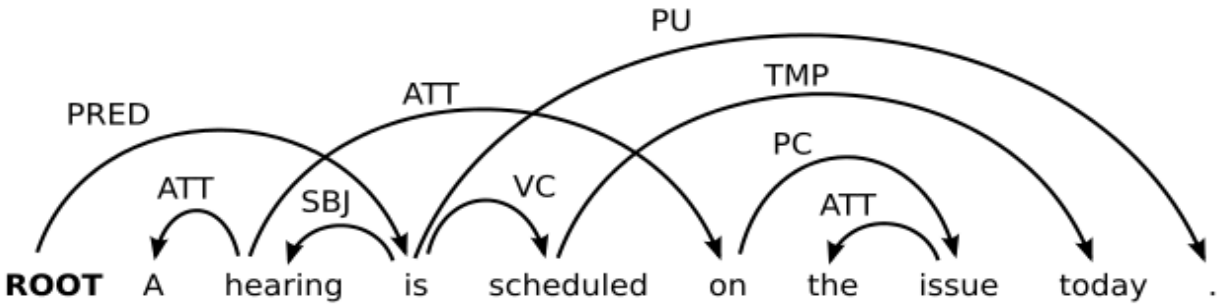
else if (σ[1] is head of σ[0] and all childs of σ[0] are attached to it and σ[0] is not the root) **then** → **Right-Arc**

else if (σ[1] > σ[0]) **then** → **SWAP**

else → **SHIFT**

This refers to the inorder traversal linear order.

Nivre's Arc-Standard with SWAP



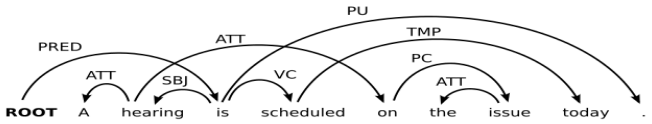
- Inorder traversal:

  A hearing **on the issue** is scheduled today

  1 2 5 6 7 3 4 8

The oracle says that we should swap whenever we have at the top of the Stack two words that break the linear order.
Example: If we would have σ[1]=**scheduled** and σ[0]=**on** we will SWAP

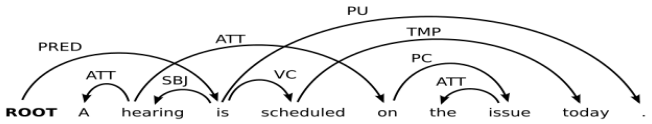
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	**Stack**	**Buffer**
	[ ]	**[A hearing is scheduled on the issue today]**

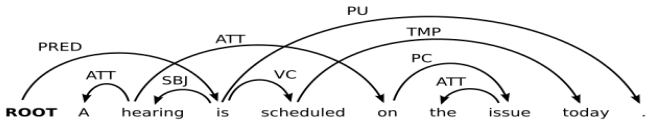
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	**Stack**	**Buffer**
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**

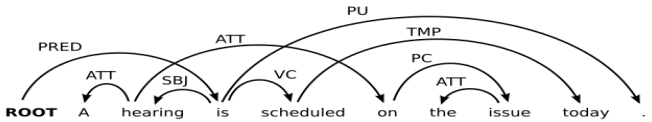
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	**Stack**	**Buffer**
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**

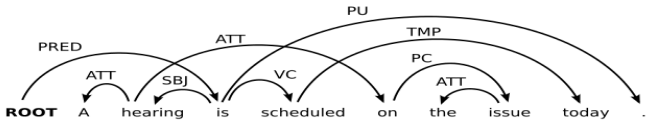
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**

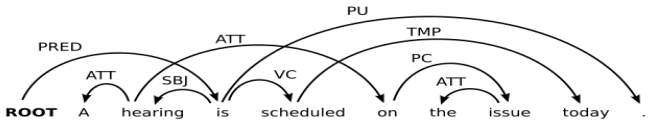
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**

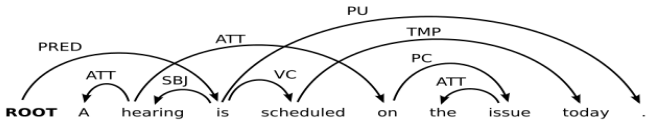
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**

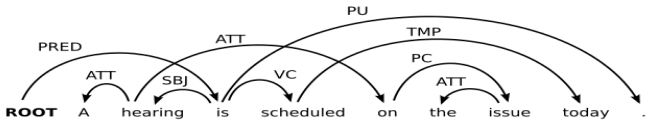
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**

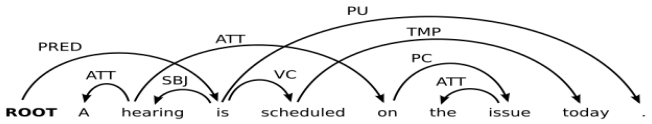
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**
SWAP	**[hearing, is, on]**	**[scheduled the issue today]**

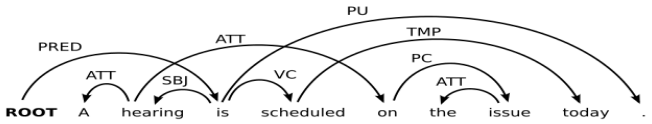
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**
SWAP	**[hearing, is, on]**	**[scheduled the issue today]**
SWAP	**[hearing, on]**	**[is, scheduled the issue today]**

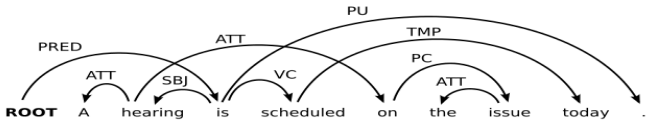
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**
SWAP	**[hearing, is, on]**	**[scheduled the issue today]**
SWAP	**[hearing, on]**	**[is, scheduled the issue today]**
SHIFT	**[hearing, on, is]**	**[scheduled the issue today]**

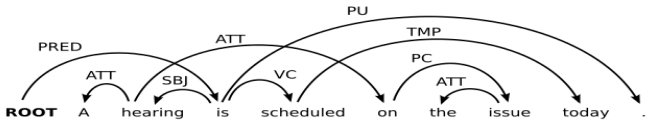
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]

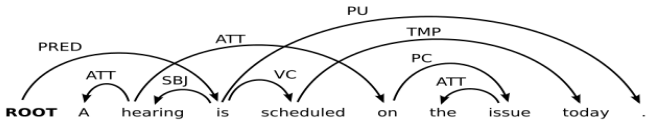
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**
SWAP	**[hearing, is, on]**	**[scheduled the issue today]**
SWAP	**[hearing, on]**	**[is, scheduled the issue today]**
SHIFT	**[hearing, on, is]**	**[scheduled the issue today]**
SHIFT	**[hearing, on, is, scheduled]**	**[the issue today]**
SHIFT	**[hearing, on, is, scheduled, the]**	**[issue today]**

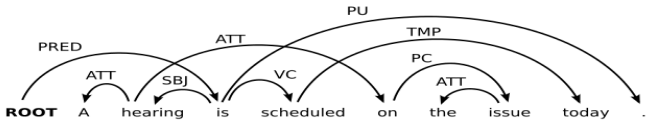
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	**[A hearing is scheduled on the issue today]**
SHIFT	**[A]**	**[hearing is scheduled on the issue today]**
SHIFT	**[A hearing]**	**[is scheduled on the issue today]**
L-A (ATT)	**[hearing]**	**[is scheduled on the issue today]**
SHIFT	**[hearing, is]**	**[scheduled on the issue today]**
SHIFT	**[hearing, is, scheduled]**	**[on the issue today]**
SHIFT	**[hearing, is, scheduled, on]**	**[the issue today]**
SWAP	**[hearing, is, on]**	**[scheduled the issue today]**
SWAP	**[hearing, on]**	**[is, scheduled the issue today]**
SHIFT	**[hearing, on, is]**	**[scheduled the issue today]**
SHIFT	**[hearing, on, is, scheduled]**	**[the issue today]**
SHIFT	**[hearing, on, is, scheduled, the]**	**[issue today]**
SWAP	**[hearing, on, is, the]**	**[scheduled, issue today]**

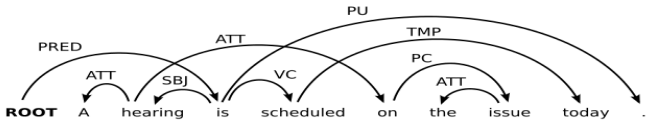
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]

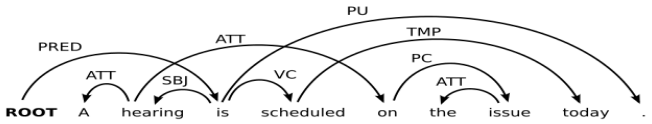
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]

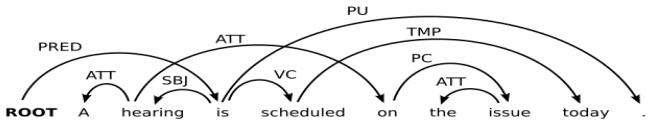
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]

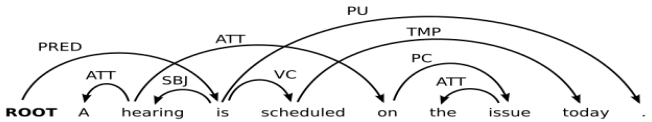
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]

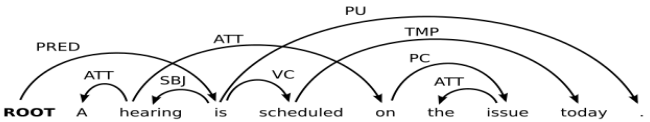
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]

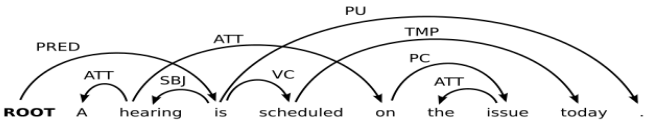
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]

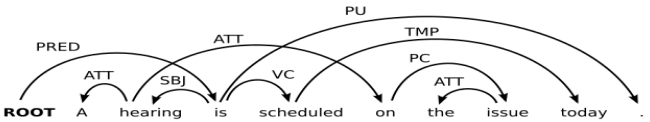
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]

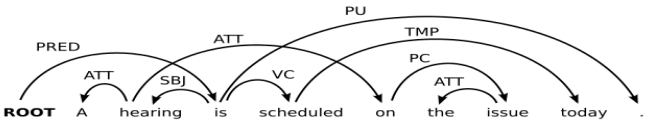
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]

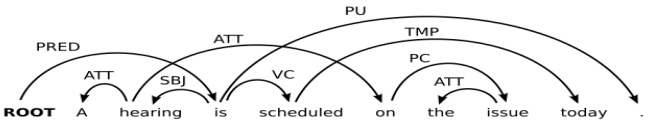
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]

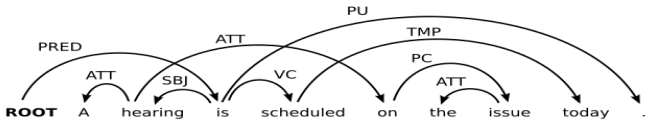
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]

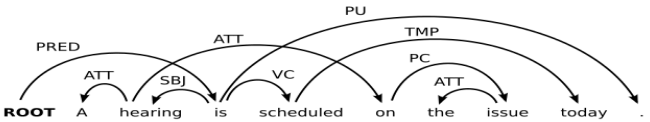
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]
LA (SBJ)	[is]	[scheduled, today]

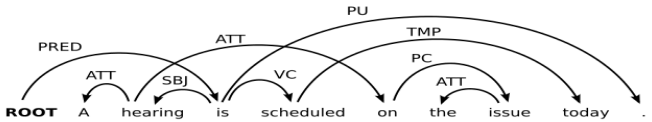
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]
LA (SBJ)	[is]	[scheduled, today]
SHIFT	[is, scheduled]	[today]

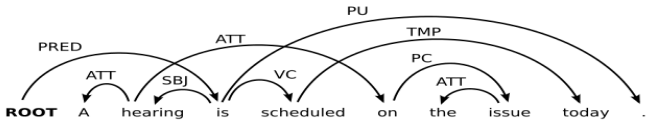
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]
LA (SBJ)	[is]	[scheduled, today]
SHIFT	[is, scheduled]	[today]
SHIFT	[is, scheduled, today]	[ ]

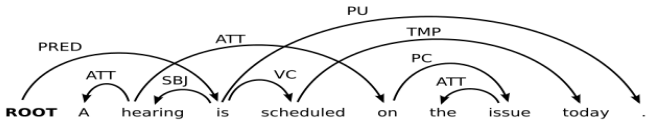
Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]
LA (SBJ)	[is]	[scheduled, today]
SHIFT	[is, scheduled]	[today]
SHIFT	[is, scheduled, today]	[ ]
RA (TMP)	[is, scheduled]	[ ]

Nivre's Arc-Standard with SWAP



A hearing **on the issue** is scheduled today
1 2 5 6 7 3 4 8

Action	Stack	Buffer
	[ ]	[A hearing is scheduled on the issue today]
SHIFT	[A]	[hearing is scheduled on the issue today]
SHIFT	[A hearing]	[is scheduled on the issue today]
L-A (ATT)	[hearing]	[is scheduled on the issue today]
SHIFT	[hearing, is]	[scheduled on the issue today]
SHIFT	[hearing, is, scheduled]	[on the issue today]
SHIFT	[hearing, is, scheduled, on]	[the issue today]
SWAP	[hearing, is, on]	[scheduled the issue today]
SWAP	[hearing, on]	[is, scheduled the issue today]
SHIFT	[hearing, on, is]	[scheduled the issue today]
SHIFT	[hearing, on, is, scheduled]	[the issue today]
SHIFT	[hearing, on, is, scheduled, the]	[issue today]
SWAP	[hearing, on, is, the]	[scheduled, issue today]
SWAP	[hearing, on, the]	[is, scheduled, issue today]
SHIFT	[hearing, on, the, is]	[scheduled, issue today]
SHIFT	[hearing, on, the, is, scheduled]	[issue today]
SHIFT	[hearing, on, the, is, scheduled, issue]	[today]
SWAP	[hearing, on, the, is, issue]	[scheduled, today]
SWAP	[hearing, on, the, issue]	[is, scheduled, today]
LA (ATT)	[hearing, on, issue]	[is, scheduled, today]
RA (PC)	[hearing, on]	[is, scheduled, today]
RA (ATT)	[hearing]	[is, scheduled, today]
SHIFT	[hearing,is]	[scheduled, today]
LA (SBJ)	[is]	[scheduled, today]
SHIFT	[is, scheduled]	[today]
SHIFT	[is, scheduled, today]	[ ]
RA (TMP)	[is, scheduled]	[ ]
RA (VC)	[is]	[ ]

Nivre's Arc-Standard with SWAP

- What is good about this?
  - We can parse non-projective trees in expected linear time.
  - See Nivre (2009) in the reading list.

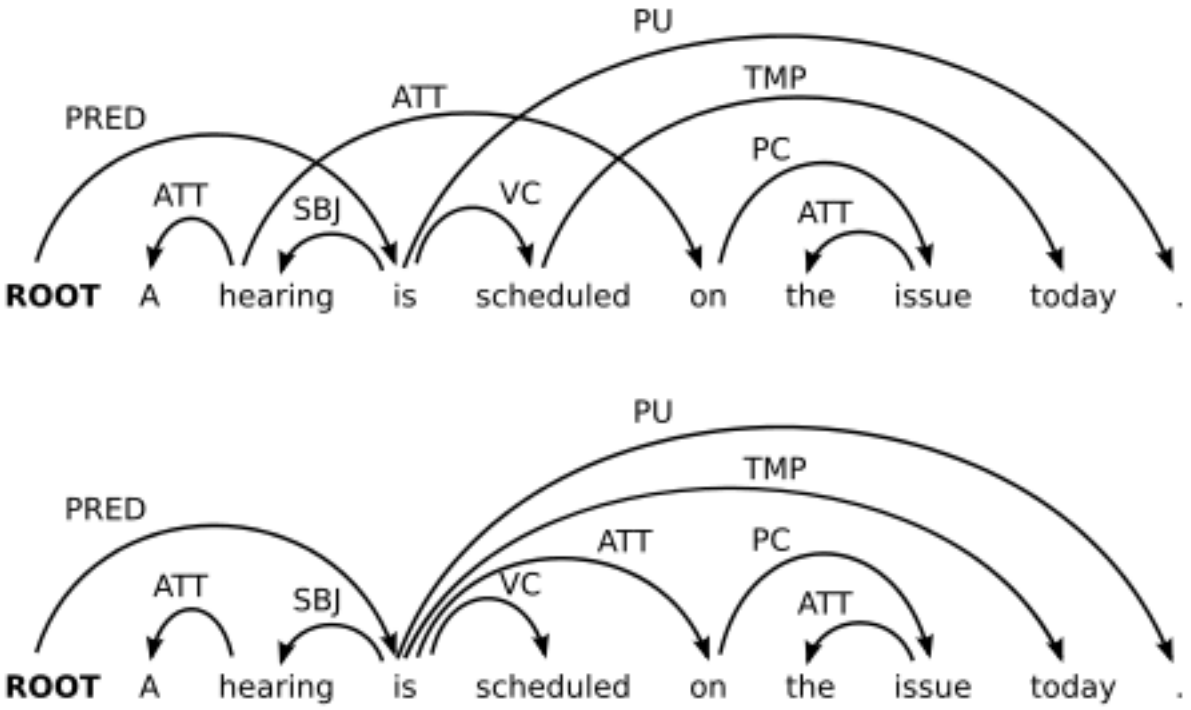
  - This technique is implemented in many parsers (among):
    - MaltParser → Greedy with SVMs.
    - Bohnet and Nivre (2012) – Beam search.
    - Bohnet et al. (2013) – Beam search.
    - Ballesteros et al. (2015) – Greedy with neural nets.

Pseudo-Projective Parsing

- The pseudo-projective transform of a non-projective tree is the tree where each non-projective arc (i, l, j) in the original tree is replaced by (k, l, j) such that k is the closest ancestor of i that does not violate the projectivity constraint. (Nivre & Nilsson, 2005)

- The idea is that we can transform any non-projective tree to a projective tree by reattaching words higher in the tree (in the worst case at the artificial root)

Pseudo-Projective Parsing



- Each non-projective arc (i, l, j) in the original tree is replaced by (k, l, j) such that k is the closest ancestor of i that does not violate the projectivity constraint.

Pseudo-Projective Parsing

- We can encode information in the labels indicating that it has been pseudo-projectivized.

- Meaning that we can “de-pseudo-projectivize” a tree, using some heuristic post-processing.
  - The output will be a non-projective tree.

Pseudo-Projective Parsing

- We take the training set and we “pseudo-projectivize” it.
- We train a projective parser (arc-std or arc-eager)
- We parse data.
- (optionally, we can “de-pseudo-projectivize” the output)

Pseudo-Projective Parsing

- This technique is implemented in many parsers. (Among others):
  - MaltParser
    - Greedy with SVMs.
  - Zpar (Yue Zhang et al.)
    - Beam search with arc-eager and perceptron learning.

Joint <many tasks> and Parsing

- Transition-based approaches to parsing are a very good fit for other (simpler) tasks.
- The typical example is joint pos tagging and parsing.

Joint Morphosyntactic tagging and Parsing

Initialization:

$c\ s\ (x = x\ 1\ ,\ .\ .\ .\ ,\ x\ n\ ) = ([0], [1, .\ .\ .\ , n], \emptyset)$

Terminal:

$Ct = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

$(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift POS-tag**)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, l, i)\})$ 1 (**Left-Arc** label)

Permitted only if $i \neq 0$.

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, l, j)\})$ (**Right-Arc** label)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], [i|\beta], A)$ 2 (**Swap**)

Permitted only if $i \neq 0$ and $i < j$.

Joint Morphosyntactic tagging and Parsing

Initialization:

$c\ s\ (x = x\ 1\ ,\ .\ .\ .\ ,\ x\ n\ ) = ([0], [1, .\ .\ .\ , n], \emptyset)$

Terminal:

$Ct = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

$(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift POS-tag**)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, l, i)\})$ 1 (**Left-Arc** label)

Permitted only if $i \neq 0$.

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, l, j)\})$ (**Right-Arc** label)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], [i|\beta], A)$ 2 (**Swap**)

Permitted only if $i \neq 0$ and $i < j$.

This can obviously be extended to other tasks such as morphological tagging, lemmatization or anything in which we want to predict an extra label per each word.

Joint Morphosyntactic tagging and Parsing

Initialization:

$c\ s\ (x = x\ 1\ ,\ .\ .\ .\ ,\ x\ n\ ) = ([0], [1, .\ .\ .\ , n], \emptyset)$

Terminal:

$Ct = \{c \in C | c = ([0], [\ ], A)\}$

Transitions:

$(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$ (**Shift POS-tag**)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], \beta, A \cup \{(j, l, i)\})$ 1 (**Left-Arc** label)

Permitted only if $i \neq 0$.

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|i], \beta, A \cup \{(i, l, j)\})$ (**Right-Arc** label)

$([\sigma|i|j], \beta, A) \Rightarrow ([\sigma|j], [i|\beta], A)$ 2 (**Swap**)

Permitted only if $i \neq 0$ and $i < j$.

This can obviously be extended to other tasks such as morphological tagging, lemmatization or anything in which we want to predict an extra label per each word.

The good thing about this is that in learning and decoding the postags and trees predicted jointly are joint so the parser learn to use the postags that is predicting for parsing, and it learns to use the syntax info for pos tagging (**history-based features!**)

Joint Morphosyntactic tagging and Parsing

The best example for this is the parser (in the reading list) by Bernd Bohnet and Joakim Nivre, that provides state-of-the-art for tagging and parsing (in some languages).

Also the parser by Alberti et al. EMNLP 2015.

- Joint pos tagging and parsing.
- Joint morphosyntactic and parsing

Joint SRL and Parsing

- This is also possible but it is harder.
- You need two stacks and play with both of them, one for the syntax and one for the semantic roles.

- (See James Henderson's parser. 2011)

Joint Disfluency Detection and Parsing

I want this project, uh I mean, this research project.

- At the same time you do parsing, you detect disfluent utterances and you mark them.
- This can be very useful to improve the output of ASR systems.
  - Rasooli and Tetrault (2013)
  - Honnibal and Johnson (2014)

Your Homework #4

- **Implement a greedy transition-based parser with arc-standard for projective trees.**
  - Train it on a subset of the penn treebank.
  - Evaluate it on a subset of the penn treebank
  - Simple classifier. Your choice. I recommend Perceptron.
  - You can use the starter code in Python.
- You are free to improve it and make it better by following some of the things I explained here (you will get better grades, if we think it is worth it).
- The better the score of your parser, the better your score in the assignment.