Language Technology
http://cs.lth.se/edan20/
Chapter 11: Syntactic Formalisms and Parsing

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

October 6, 2022

Syntax

Syntax has been the core of linguistics in the US and elsewhere for many years

Noam Chomsky, professor at the MIT, has had an overwhelming influence, sometimes misleading

Syntactic structures (1957) has been a cult book for the past generation of linguists

Syntax can be divided into two parts:

- Formalism – How to represent syntax
- Parsing – How to get the representation of a sentence

Syntactic Formalisms

The two most accepted formalisms use a tree representation:

- One is based on the idea of constituents
- Another is based on dependencies between words. Trees have originally been called stemmas

They are generally associated respectively to Chomsky and Tesnière.

Later, constituent grammars evolved into unification grammars

Constituency

Constituency can be expressed by context-free grammars. They are defined by

1. A set of designated start symbols, Σ, covering the sentences to parse. This set can be reduced to a single symbol, such as `sentence`, or divided into more symbols: `declarative_sentence`, `interrogative_sentence`.
2. A set of nonterminal symbols enabling the representation of the syntactic categories. This set includes the sentence and phrase categories.
3. A set of terminal symbols representing the vocabulary: words of the lexicon, possibly morphemes.
4. A set of rules, F, where the left-hand-side symbol of the rule is rewritten in the sequence of symbols of the right-hand side.

DCG

These grammars can be mapped to DCG rules as for

The boy hit the ball

```
sentence --> np, vp.
np --> t, n.
vp -- verb, np.
t --> [the].
n --> [man] ; [ball] ; etc.
verb --> [hit] ; [took] ; etc.
```

Generation of sentences is one of the purposes of grammar according to Chomsky

Chomsky Normal Form

In some parsing algorithms, it is necessary to have rules in the Chomsky normal form (CNF) with two right-hand-side symbols
Non-CNF rules:

```
lhs --> rhs1, rhs2, rhs3.
```

can be converted into a CNF equivalent:

```
lhs --> rhs1, lhs_aux.
lhs_aux --> rhs2, rhs3.
```

Transformations

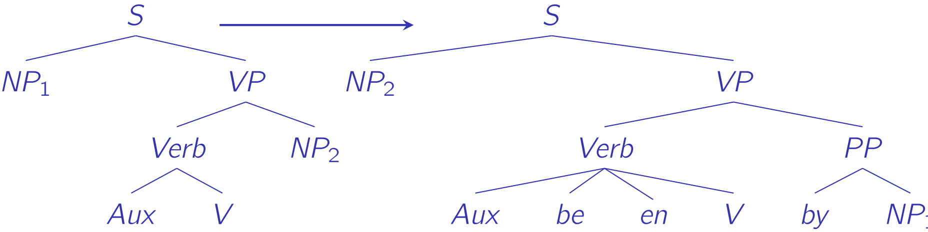
Rearrangement of sentences according to some syntactic relations: active/passive, declarative/interrogative, etc.

Transformations use rules – transformational rules or T rules –

The boy will hit the ball/the ball will be (en) hit by the boy

```
T1:  np1, aux, v, np2 --->
     np2, aux, [be], [en], v, [by], np1
```

Transformations



Syntactic Categories (Penn Treebank)

	Categories	Description
1.	ADJP	Adjective phrase
2.	ADVP	Adverb phrase
3.	NP	Noun phrase
4.	PP	Prepositional phrase
5.	S	Simple declarative clause
6.	SBAR	Clause introduced by subordinating conjunction or 0
7.	SBARQ	Direct question introduced by *wh*-word or phrase
8.	SINV	Declarative sentence with subject-aux inversion
9.	SQ	Subconstituent of SBARQ excluding *wh*-word or phras
10.	VP	Verb phrase
11.	WHADVP	*wh*-adverb phrase
12.	WHNP	*wh*-noun phrase
13.	WHPP	*wh*-prepositional phrase
14.	X	Constituent of unknown or uncertain category

A Hand-Parsed Sentence using the Penn Treebank Annotation

Battle-tested industrial managers here always buck up nervous newcomers with the tale of the first of their countrymen to visit Mexico, a boatload of samurai warriors blown ashore 375 years ago.

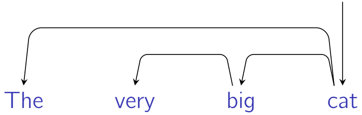
```
( (S
    (NP Battle-tested industrial managers
         here)
    always
    (VP buck
         up
         (NP nervous newcomers)
         (PP with
              (NP the tale
              (PP of
```

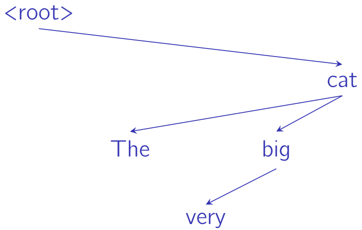
A Hand-Parsed Sentence using the Penn Treebank Annotation

```
(NP (NP the
        (ADJP first
              (PP of
                  (NP their countrymen)))
        (S (NP *)
           to
           (VP visit
               (NP Mexico))))
    ,
    (NP (NP a boatload
            (PP of
                (NP (NP samurai warriors)
                    (VP-1 blown
                        ashore
                    (ADVP (NP 375 years)
                          ago)))))
        (VP-1 *pseudo-attach*)))))))))
```

Dependency Grammars

Dependency grammars (DG) describe the structure in term of links





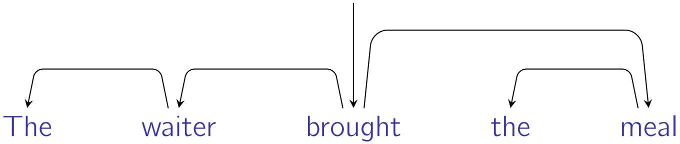
Each word has a head or “régissant” except the root of the sentence.

A head has one or more modifiers or dependents:

Cat is the head of *big* and *the*; *big* is the head of *very*.

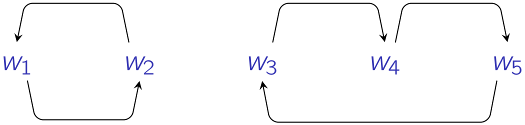
DG can be more versatile with a flexible word order language like German, Russian, or Latin.

A Sentence Tree – Stemma

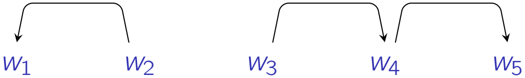


Properties of Dependency Graphs

Acyclic

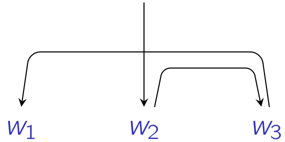


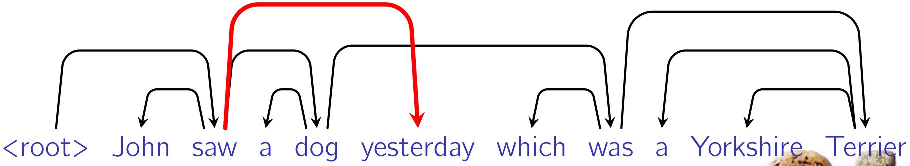
Connected



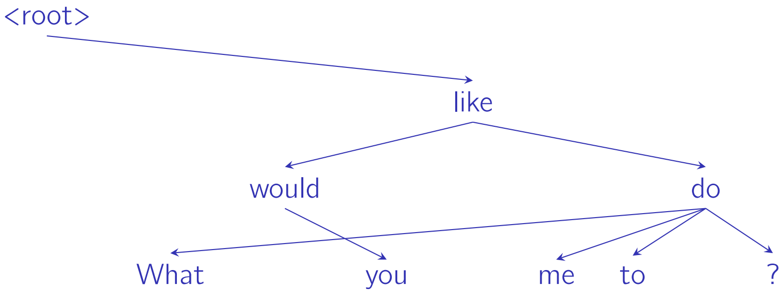
Projective Each pair of words (Dep, Head), directly connected, is only separated by direct or indirect dependents of Dep or Head

Nonprojective Graphs (McDonald and Pereira)





Nonprojective Graphs (Järvinen and Tapanainen)



Valence

Tesnière makes a distinction between essential and circumstantial complements

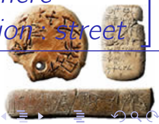
Essential – or core – complements are for instance subject and objects.

Circumstantial – or noncore – complements are the adjuncts

Valence corresponds to the verb saturation of its essential complements

Valence Examples

Val.	Examples	Frames
0	*it's raining*	*raining* $[]$
1	*he's sleeping*	*sleeping* $[\textit{subject} : \textit{he}]$
2	*she read this book*	*read* $\begin{bmatrix} \textit{subject} : \textit{she} \\ \textit{object} : \textit{book} \end{bmatrix}$
3	*Elke gave a book to Wolfgang*	*gave* $\begin{bmatrix} \textit{subject} : \textit{Elke} \\ \textit{object} : \textit{book} \\ \textit{iobject} : \textit{Wolfgang} \end{bmatrix}$
4	*I moved the car from here to the street*	*moved* $\begin{bmatrix} \textit{subject} : \textit{I} \\ \textit{object} : \textit{car} \\ \textit{source} : \textit{here} \\ \textit{destination} : \textit{street} \end{bmatrix}$



Subcategorization Frames

Valence is a model of verb construction. It can be extended to more specific patterns as in the *Oxford Advanced Learner's Dictionary* (OALD).

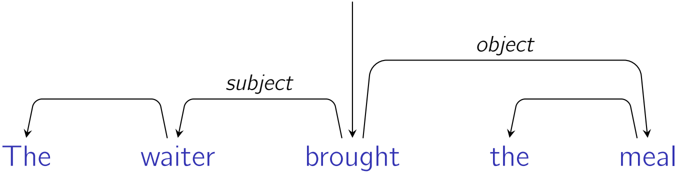
Verb	**Complement structure**	**Example**
slept	None (Intransitive)	*I slept*
bring	NP	*The waiter brought the meal*
bring	NP + to + NP	*The waiter brought the meal to the patron*
depend	on + NP	*It depends on the waiter*
wait	for + NP + to + VP	*I am waiting for the waiter to bring the meal*
keep	VP(ing)	*He kept working*
know	that + S	*The waiter knows that the patron loves fish*

Dependencies and Grammatical Functions

The dependency structure generally reflects the traditional syntactic representation

The links can be annotated with grammatical function labels.

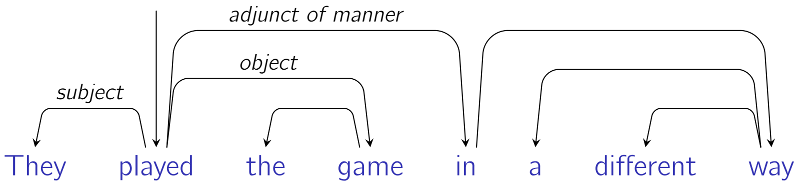
In a simple sentence, it corresponds to the subject and the object



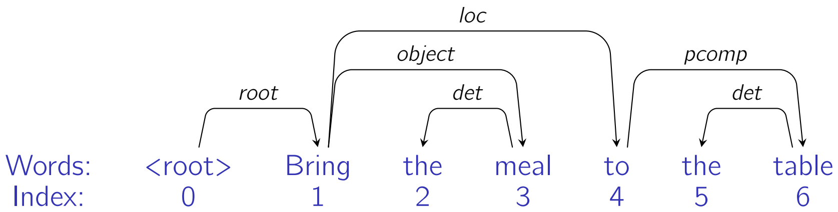
Probably a more natural description to tie syntax to semantics

Dependencies and Functions (II)

Adjuncts form another class of functions that modify the verb
They include prepositional phrases whose head is set arbitrarily to the front preposition
Adjuncts include adverbs that modify a verb



Dependency Parse Tree



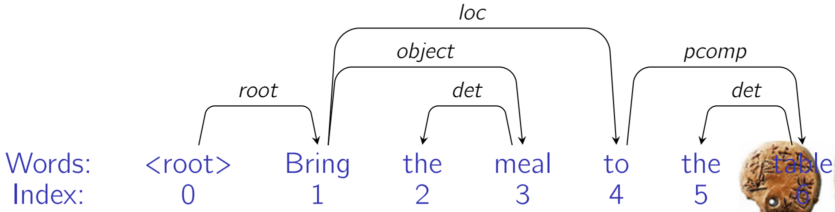
Word pos.	Word	Direction	Head	Head position	Function
1	*Bring*	*		Root	Main verb
2	*the*	>	*meal*	3	Determiner
3	*meal*	<	*Bring*	1	Object
4	*to*	<	*Bring*	1	Location
5	*the*	>	*table*	6	Determiner
6	*table*	<	*to*	4	Prepositional complement

Representing Dependencies

$$D = \{< \mathrm{Head}(1), \mathrm{Rel}(1) >, < \mathrm{Head}(2), \mathrm{Rel}(2) >, ..., < \mathrm{Head}(n), \mathrm{Rel}(n) >\},$$

The representation of *Bring the meal to the table*:

$$D = \{< 0, \mathrm{root} >, < 3, \mathrm{det} >, < 1, \mathrm{object} >, < 1, \mathrm{loc} >, < 6, \mathrm{det} >, < 4, \mathrm{pcomp} >\},$$



Annotation: CoNLL-X

The CoNLL shared tasks organize evaluations of machine-learning systems for natural language processing.
They define formats to share data between participants.

1	Dessutom	_	AB	AB	_	2	+A	_	_
2	höjs	_	VV	VV	_	0	ROOT	_	_
3	åldergränsen	_	NN	NN	_	2	SS	_	_
4	till	_	PR	PR	_	2	OA	_	_
5	18	_	RO	RO	_	6	DT	_	_
6	år	_	NN	NN	_	4	PA	_	_
7	.	_	IP	IP	_	2	IP	_	_

Annotation: CoNLL-U (simplified)

CoNLL-U is an attempt to unify the grammatical annotation across human languages.

ID	FORM	LEMMA	UPOS	HEAD	DEPREL
1	Dessutom	dessutom	ADV	2	advmod
2	höjs	höja	VERB	0	root
3	åldergränsen	åldergräns	NOUN	2	nsubj:pass
4	till	till	ADP	6	case
5	18	18	NUM	6	nummod
6	år	år	NOUN	2	obl
7	.	.	PUNCT	2	punct

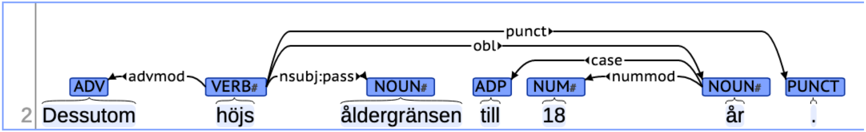
Corpora available in many languages:
https://universaldependencies.org/

Annotation: CoNLL-U

#	Name	Description
1	ID	Word index, integer starting at 1 for each new sentence; may be a range for tokens with multiple words.
2	FORM	Word form or punctuation symbol.
3	LEMMA	Lemma or stem of word form.
4	UPOS	Universal part-of-speech tag.
5	XPOS	Language-specific part-of-speech tag; underscore if not available.
6	FEATS	List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available).
7	HEAD	Head of the current token, which is either a value of ID or zero (0).
8	DEPREL	Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
9	DEPS	Enhanced dependency graph in the form of a list of head-deprel pairs.
10	MISC	Any other annotation.

Visualizing Dependencies

Using *conllu.js* (`http://spyysalo.github.io/conllu.js/`):



Function Annotation Tagset (Järvinen and Tapanainen 1997)

Name	Description	Example
	Main functions	
main	Main element	*He doesn't* **know** *whether to send a gift*
qtag	Question tag	*Let's play another game,* **shall** *we?*
	Intranuclear links	
v-ch	Verb chain	*It* **may have been being** *examined*
pcomp	Prepositional complement	*They played the game* **in** *a different way*
phr	Verb particle	*He asked me who would look* **after** *the baby*

Function Annotation Tagset (Järvinen and Tapanainen 1997)

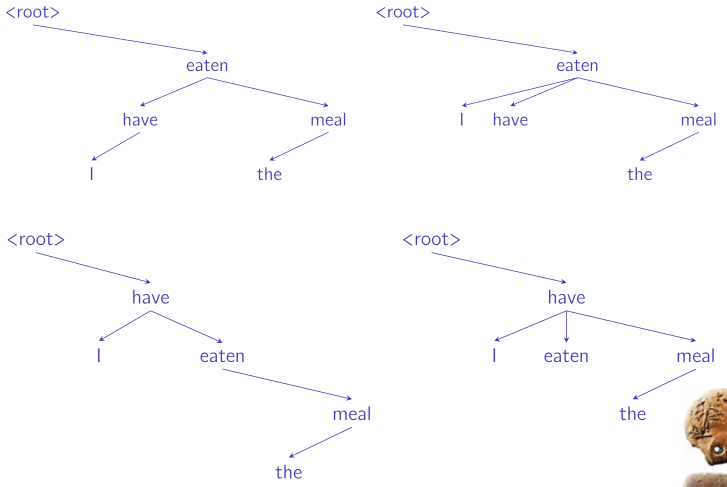
	Verb complementation	
subj	Subject	
obj	Object	*I gave him my* **address**
comp	Subject complement.	*It has become* **marginal**
dat	Indirect object	*Pauline gave it* **to** *Tom*
oc	Object complement	*His friends call him* **Ted**
copred	Copredicative	*We took a swim* **naked**
voc	Vocative	*Play it again*, **Sam**
	Determinative functions	
qn	Quantifier	*I want* **more** *money*
det	Determiner	**Other** *members will join…*
neg	Negator	*It is* **not** *coffee that I like, but tea*

Function Annotation Tagset (Järvinen and Tapanainen 1997)

	Modifiers	
attr	Attributive nominal	***Knowing*** *no French, I couldn't express my thanks*
mod	Other postmodifiers	*The baby, Frances* ***Bean****, was...*
		The people ***on*** *the bus were singing*
ad	Attributive adverbial	*She is* ***more*** *popular*
	Junctives	
cc	Coordination	*Two* ***or*** *more cars...*

Differences in Annotation Conventions

Dependency graph of *I have eaten the meal* with different conventions to relate an auxiliary to its main verb



Dependency vs. Constituency

Constituency (most textbooks) is a declining formalism

It cannot properly handle many languages: Swedish, Russian, Czech, Arabic, etc.

Dependency parsing can handle all these languages as well as English, German, French, etc.

Dependency parsing has improved considerably over the last 10 years: see CoNLL 2006 and 2007, then CoNLL 2017 and 2018.

CoNLL 2008 and 2009 extended it to semantic parsing

However, constituency and dependency are (weakly) compatible provided that we restrict us to projective dependency graphs

From Constituency to Dependency

It is possible to convert constituent trees into dependency graphs

We need to identify a headword in all the PS rules, here with a star:

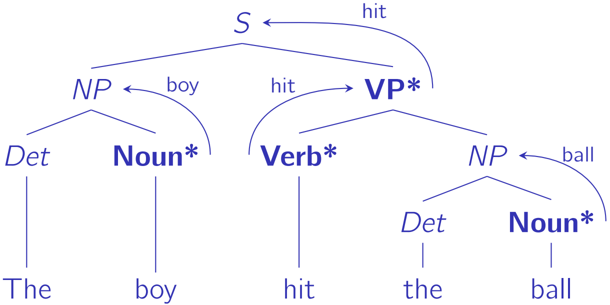
```
s --> np, vp*.
vp --> verb*, np.
np --> det, noun*.
```

Parsers by Magerman and Collins used this to convert the Penn Treebank constituent annotation for their dependency parsers

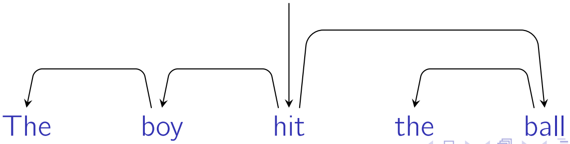
When projective, dependency structures are loosely compatible with constituent grammars.

From Constituency to Dependency (II)

A constituent tree with head-marked rules:



The resulting dependency graph:



Parsing Dependencies

Generate all the pairs:

Which sentence root?

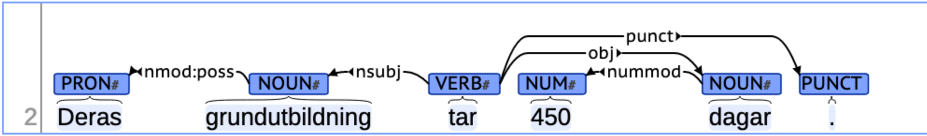
Bring
the
meal
to
the
table

Which head?
meal

Talbanken: An Annotated Corpus of Swedish

1	Deras	de	PRON	2	nmod:poss
2	grundutbildning	grundutbildning	NOUN	3	nsubj
3	tar	ta	VERB	0	root
4	450	450	NUM	5	nummod
5	dagar	dag	NOUN	3	obj
6	.	.	PUNCT	3	punct

Visualizing the Graph



Parser Input

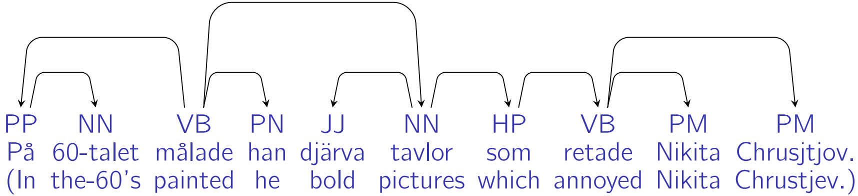
The words and their parts of speech obtained from an earlier step.

1	Deras	de	PRON
2	grundutbildning	grundutbildning	NOUN
3	tar	ta	VERB
4	450	450	NUM
5	dagar	dag	NOUN
6	.	.	PUNCT

Nivre's Parser

Joakim Nivre designed an efficient dependency parser extending the shift-reduce algorithm.

He started with Swedish and has reported the best results for this language and many others.



His team obtained the best results in the CoNLL 2007 shared task on dependency parsing.

The Parser (Arc-Eager)

The first step is a POS tagging

The parser applies a variation/extension of the shift-reduce algorithm since dependency grammars have no nonterminal symbols

The transitions are:

1. **Shift**, pushes the input token onto the stack
2. **Right arc**, adds an arc from the token on top of the stack to the next input token and pushes the input token onto the stack.
3. **Reduce**, pops the token on the top of the stack
4. **Left arc**, adds an arc from the next input token to the token on the top of the stack and pops it.

Transitions' Definition

We use a triple: $\langle S, I, A\rangle$, where S is the stack, I, the input list, and A, the set of arcs in the graph.

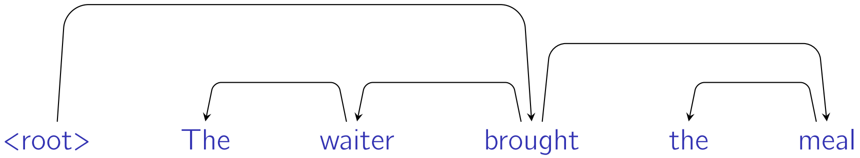
Actions	Parser states	Conditions
Initialization	$\langle nil, W, \emptyset\rangle$	
Termination	$\langle S, [], A\rangle$	
Shift	$\langle S, [n\|I], A\rangle \rightarrow \langle [S\|n], I, A\rangle$	
Reduce	$\langle [S\|n], I, A\rangle \rightarrow \langle S, I, A\rangle$	$\exists n'(n', n) \in A$
Left-arc	$\langle [S\|n], [n'\|I], A\rangle \rightarrow \langle S, [n'\|I], A \cup \{(n \leftarrow n')\}\rangle$	$\nexists n''(n'', n) \in A$
Right-arc	$\langle [S\|n], [n'\|I], A\rangle \rightarrow \langle [S\|n, n'], I, A \cup \{(n \rightarrow n')\}\rangle$	

1. Left-arc is an augmented reduce, and right-arc, an augmented shift.
2. The first condition $\exists n'(n', n) \in A$, where n' is the head and n, the dependent, is to ensure that the graph is connected.
3. The second condition $\nexists n''(n'', n) \in A$, where n'' is the head and n, the dependent, is to enforce a unique head.

Nivre's Parser in Action

Input W = The waiter brought the meal.

The graph is:



$$\{\text{the} \leftarrow \text{waiter}, \text{waiter} \leftarrow \text{brought}, \text{ROOT} \rightarrow \text{brought}, \text{the} \leftarrow \text{meal},$$
$$\text{brought} \rightarrow \text{meal}\},$$

Let us apply the sequence:

`[sh, sh, la, sh, la, ra, sh, la, ra]`

Nivre's Parser in Action

[sh, sh, la, sh, la, ra, sh, la, ra]

Trans.	Stack	Queue	Graph
start	$\emptyset$	[ROOT, the, waiter, brought, the, meal]	{}
sh			
	[ROOT]	[the, waiter, brought, the, meal]	{}
sh			
	[the ROOT]	[waiter, brought, the, meal]	{}
la			
	[ROOT]	[waiter, brought, the, meal]	{the $\leftarrow$ waiter}
sh			
	[waiter ROOT]	[brought, the, meal]	{the $\leftarrow$ waiter}
la			
	[ROOT]	[brought, the, meal]	{the $\leftarrow$ waiter, waiter $\leftarrow$ brought}

Nivre's Parser in Action (II)

[~~sh, sh, la, sh, la~~, ra, sh, la, ra]

Trans.	Stack	Queue	Graph
ra			
	[brought ROOT]	[the, meal]	{the ← waiter, waiter ← brought, ROOT → brought}
sh			
	[the brought ROOT]	[meal]	{the ← waiter, waiter ← brought, ROOT → brought}
la			
	[brought ROOT]	[meal]	{the ← waiter, waiter ← brought, ROOT → brought, the ← meal}
ra			
end	[meal brought ROOT]	[]	{the ← waiter, waiter ← brought, ROOT → brought, the ← meal, brought → meal}