

TSM AnTeDe

Understanding Syntax

Daniele Puccinelli

University of Applied Sciences and Arts of Southern Switzerland (SUPSI)

Syntactic Parsing = mapping the syntactic structure of sentences

Why?

- Grammar checking: if you can't parse a sentence, there must be something wrong with it
- Intermediate stage of representation for semantic analysis
 - information extraction
 - question answering

Formal grammar: set of rules used to build syntactically valid sentences

- Nearly all natural languages seem to follow the Context-Free Grammar model
- Swiss-German is a notable exception: it needs a mildly context-sensitive grammar¹

¹S. Shieber, Evidence Against the Context-Freeness of Natural Language, Linguistics and Philosophy, Vol. 8, 1985

A CFG includes:

- a set of terminals T : words you see on the page
- a set of non-terminals N : labels we give to the words and groups of words
- a designated start symbol $S \in N$
- a set of rules (also called *productions*) of the form $A \rightarrow \beta$
 - A is a non-terminal (LHS = Left-Hand Side)
 - β is a sequence of terminals and non-terminals (RHS = Right-Hand Side)
 - $A \rightarrow \beta$ means **replace A with β**

Why do we say context-free?

- Rules can be applied regardless of context
- When you apply a rule to a symbol, nothing matters other than the symbol you apply the rule to

- T : {dogs, play, cuddly, wag}
- N : {Sentence, Noun, Verb, Adjective, NounPhrase, VerbPhrase}
- S =Sentence
- the rules are:
 - Sentence \rightarrow NounPhrase
 - Sentence \rightarrow NounPhrase VerbPhrase
 - NounPhrase \rightarrow Noun
 - NounPhrase \rightarrow Adjective Noun
 - VerbPhrase \rightarrow Verb
 - Noun \rightarrow dogs
 - Adjective \rightarrow cuddly
 - Verb \rightarrow {play, wag}

To create a legal string:

- Begin the string with the start symbol Sentence
- Apply a production rule
- Repeat until you have a string of terminals

- Sentence \rightarrow NounPhrase
- NounPhrase \rightarrow Adjective Noun
- Adjective \rightarrow cuddly
- Noun \rightarrow dogs
- Our string is *cuddly dogs*
- Sentence \rightarrow NounPhrase Verb Phrase
- NounPhrase \rightarrow Noun
- VerbPhrase \rightarrow Verb
- Noun \rightarrow dogs
- Verb \rightarrow wag
- Our string is *dogs wag*

Smaller units combine into increasingly complex ones

- phrase: a group of words standing together as a conceptual unit
 - Noun Phrase: a phrase built around a noun (e.g.: the dog, the smart dog, the very clever dog)
 - Verb Phrase: a phrase built around a verb (e.g.: went home, is on the couch, are nice)
 - Prepositional Phrase: a phrase including a preposition and a noun phrase that indicates relations in space and time (e.g., at home, after dinner)
 - Adjective/Adverb Phrase: a phrase built around adjectives and adverbs (e.g., very politically correct, which is an adjective phrase including the adverb phrase *very politically* that qualifies the adjective *correct*)

- clause: includes at least a subject and a verb (e.g.: I'm going home; e.g.: because I'm tired)
- sentence: a group of clauses
 - simple sentence: only an independent clause (e.g.: I'm coming home)
 - compound sentence: at least two independent clauses (e.g.: I'm coming home but I'm not watching your show)
 - complex sentence: one independent clause and one or more dependent clauses (e.g.: I'm coming home because I'm tired)
 - compound-complex sentence: two or more independent clauses and one or more dependent clauses (e.g.: I'm coming home but I'm not watching your show because I'm tired).

A CFG represents how words and phrases are organized in a sentence

If a sentence is part of the language defined by a CFG, we know how its constituents (its parts) are organized

- In practice, we need a probabilistic CFG (PCFG)
- The same as a CFG but every rule has a certain probability
- All probabilities for the same LHS must sum to 1
- We can learn these probabilities from treebanks!

We can combine words into phrases based on CFG rules

- Say we want to build a Noun Phrase (NP)
- We may apply a rule $NP \rightarrow DT\ NN$ to build a NP, like *the dog*
- We may apply $NP \rightarrow DT\ (JJ^1)\ NN$ and enrich our Noun Phrase with an (optional) adjective: *the {big | lovely | cuddly} dog*
- We may build a prepositional phrase by applying $PP \rightarrow IN^2\ NP$: *on the table | by the door | in my arms*
- Now we build a more complex NP with the rule $NP \rightarrow DT\ (JJ)\ NN\ (PP)$: e.g., *the cuddly dog in my arms*
- We can go on with other rules and combine phrases into sentences

What we do in practice is the opposite: we see how words are organized based on CFG rules

¹JJ = adjective in the Penn tagset

²IN = preposition in the Penn tagset

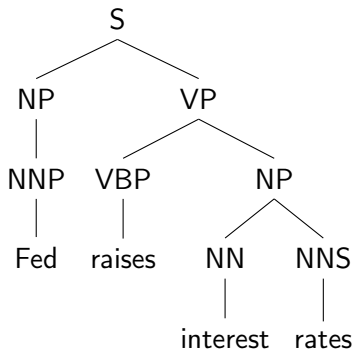
Example: *Fed raise*

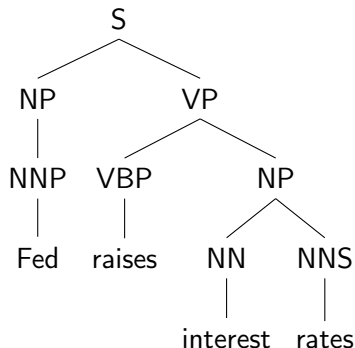
- With PoS tagging, we identify the individual parts of speech

NNP	VBP	NN	NNS
Fed	raises	interest	rates

Example: *Fed raises*

- We can go up one layer and identify noun phrases (NP), verb phrases (VP), and the whole sentence (S)





- *interest* and *rates* go together as a noun phrase
- *raises* and *interest rates* go together as a verb phrase
- *Fed* and *raises interest rates* go together as a sentence
- Each NP and VP unit is a **constituent** (a part of the sentence)
- Constituents can be indicated with brackets: Fed [raises [interest rates]]

Try everything yourself in Notebook A as I walk you through the lecture

jupyter MSE_AnTeDe_Lab12_ParsingBasics (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run

```
In [3]: 1 s = StanfordNLP()
        2
        3 sentence='Fed raises interest rates.'
        4
        5 r=s.parse(sentence)
```

ROOT

S

NP VP

NNP VBZ NN NNS .

Fed raises interest rates .

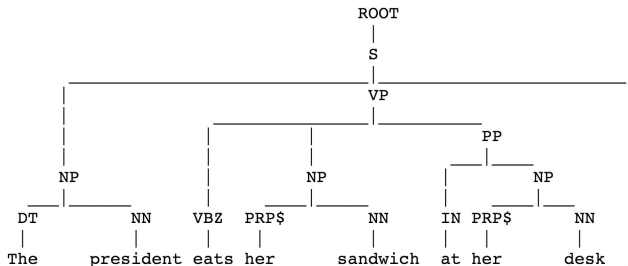
- You're less likely to fall asleep
- You'll probably have more fun

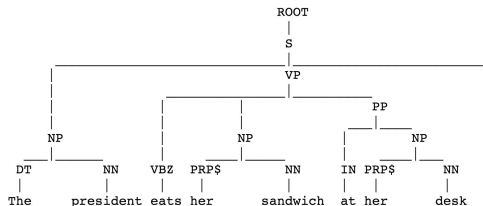
How do we know what a constituent is?

- it's a complex question and linguists often disagree, but there are some basic tests
- look for words that stay together when phrases move around
 - Mary talked **to her daughters** **about Cinderella** → OK
 - Mary talked **about Cinderella** **to her daughters** → OK
 - **To her daughters**, Mary talked **about Cinderella** → OK
 - * Mary talked **Cinderella** **to her daughters** **about** → NO!
- phrasal expansion and substitution
 - I sat [on the box]
 - I sat [on the box | right on top of the box | on top of it | there]
- Lots of other tests

Try it yourself using Stanford CoreNLP's parser in Lab A!

```
In [3]: 1 s = StanfordNLP()  
        2 sentence_1='The president eats her sandwich at her desk.'  
        3  
        4 r=s.parse(sentence_1)  
        5
```





- The president is a NP¹
- her sandwich is a NP
- at her desk is a PP²
- eats her sandwich at her desk is a VP³ → VBZ NP PP
- You get the whole sentence with the rule S⁴ → NP VP

¹noun phrase

²prepositional phrase

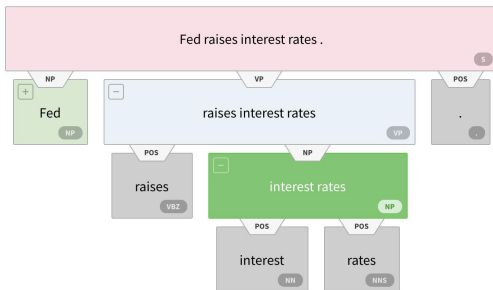
³verb phrase

⁴sentence

- S - simple declarative clause (e.g., I'm taking this class)
- SBAR - clause introduced by a (possibly empty) subordinating conjunction (e.g. I think I'm stupid; I think that I'm stupid)
- SBARQ - direct questions introduced by a wh-word or a wh-phrase (e.g. What do you mean?)
- SINV - Inverted declarative sentence (e.g., Never has so much been owed ...)
- SQ - Yes/No questions (e.g. Do you live here?)

This was constituency parsing

Check out the AllenNLP demos based on ELMo (Embeddings from Language Models)¹

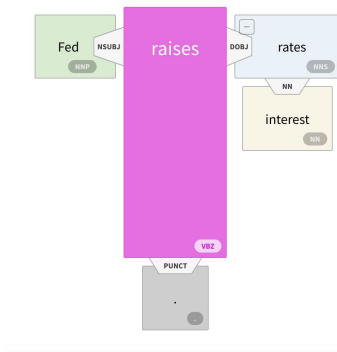


Try this yourself at
<https://demo.allennlp.org/constituency-parsing>

¹M. Peters et al., Deep contextualized word representations, NAACL 2018

The alternative is dependency parsing

Basic idea: every word is another word's dependent, except for the root



The root corresponds to the central idea of the sentence

Example: *The pre*

- The central idea is *eating*
- *eat* has three arguments:
 - 1 *The president*, who performs the action of eating (**who?**) → **nsubj**
 - *president* gets modified by *the* (article/determiner)
 - 2 *sandwich*, the thing that gets eaten (**what?**) → **dobj**¹
 - *sandwich* gets modified by *her* (**whose?**) → **nmod**²
 - 3 *desk*, the location where the action of eating takes place (**where?**) → **nmod**
 - *at* complements *desk* → **case**³
 - *her* modifies *desk* → **nmod**⁴

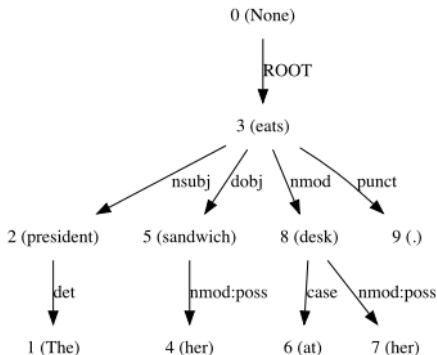
¹direct object

²nominal modifier

³case marking; in English it's used for any preposition

⁴nominal modifier

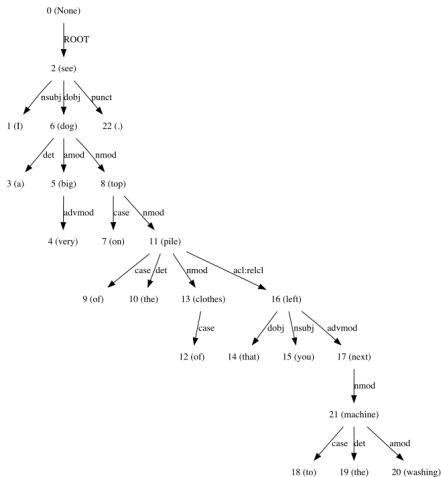
The pre



I see a very big dog on to
next to the washing machine.

- The central idea is *seeing*
- see has two arguments:
 - ① I, the person who's *seeing* → **nsubj**
 - ② dog, which is what I'm *seeing* (**what?**) → **dobj**
 - If we stop here, we get *I see dog.*, which is ungrammatical but conveys the gist of the sentence
- dog is complemented/modified by:
 - ① a, which serves as a determiner of *dog* → **det**
 - ② big, which serves as an adjectival modifier of *dog* → **amod**
 - very acts as an adverbial modifier of *big* → **advmod**
 - ③ top, which is where the *dog* is (**where?**) and acts as a nominal modifier → **nmod**
 - So far, we have *I see a big dog top.*
 - Not the Queen's English, but you get the idea.

Dependency parsing - example 2



I see a very big dog on top next to the washing machine.

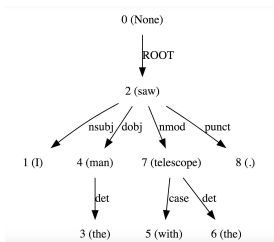
- *top* is complemented/modified by:
 - ① *on*, which complements *top* as a case marking → **case**
 - ② *pile*, which serves as a noun modifier of *top* → **nmod**
 - So far, *I see a very big dog on top pile.*
- *pile* is complemented/modified by:
 - ① *of* from *on top of the*, which complements *pile* as a case marking → **case**
 - ② *the* from *on top of the*, which complements *pile* as a determiner → **det**
 - ③ *clothes*, which serves as a noun modifier of *pile* → **nmod**
 - *clothes* is complemented by *of* (from *of clothes*) as a case marking → **case**
 - ④ *left*, which serves as a relative clause modifier of *pile* → **acl:relc**
 - You left the clothes, not the pile, so this is technically incorrect!!!
 - So far, *I see a very big dog on top of the pile clothes left.*

I see a very big dog on to
next to the washing machine.

- *left* has three arguments:
 - ① *that*, which acts as a relative pronoun standing for the *clothes* (or the *pile*, according to our parse), which is the direct object you left → **dobj**
 - ② *you*, who left them → **nsubj**
 - ③ *next*, which is where you left them → **advmod**
 - So far, *I see a very big dog on top of the pile of clothes that you left next.*
- *machine* is complemented/modified by:
 - ① *to*, which complements *machine* as a case marking → **case**
 - ② *the*, which complements *machine* as a determiner → **det**
 - ③ *washing*, which acts as an adjective modifier → **amod**
 - We're done: *I see a very big dog on top of the pile of clothes that you left next to the washing machine.*

Example: I saw the man with a tele

- *saw* is the root
- *I* and *man* depend on the root
- If I used a telescope to see the man, then *telescope* also depends on the root
- If the man was holding a telescope, then *telescope* depends on *man*



What does CoreNLP think?

By Jonathan Amos
BBC Science Correspondent

We will look into the details of some key algorithms:

Constituency Parsing: how are the sentence elements organized?

- Probabilistic Cocke-Kasami-Younger (CKY) parser

Dependency Parsing: what depends on what?

- Arc-standard transition-based parser
- Neural dependency parser

- At any given time, the parser has a configuration:
 - A **stack** of words (top to the right)
 - A **buffer** of words (top to the left)
 - A (partial) **dependency graph**
- The parser begins in an **initial configuration**
 - **stack**: ROOT symbol
 - **buffer**: all words
 - **dependency graph**: empty

The possible actions are:

- 1 SHIFT: move a word from the top of the buffer to the top of the stack
- 2 LA_r : Left Arc
 - the neighbor of the head of the **stack** is a dependent of the head of the **stack**
 - add a corresponding link to the **dependency graph** with label r
 - remove the neighbor from the **stack**
- 3 RA_r : Right Arc
 - the head of the **stack** is a dependent of its neighbor
 - add a corresponding link to the **dependency graph** with label r
 - remove the head of the **stack**

Example¹: parse the sentence *I ate fish*

Initial configuration: the stack contains the *root* symbol, the buffer contains the sentence



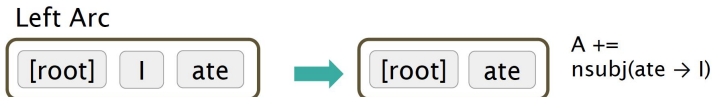
- The first step is a SHIFT because *I* is not the root
- The second step is also a SHIFT because *ate* doesn't depend on *I*



- *I* depends on the root *ate* (**nsubj**), so the third step will be a ...

¹From Christopher Manning, CS224N/Ling284 2017, Stanford University

- *I* depends on the root *ate* (**nsubj**), so the third step will be a LA_{nsubj}



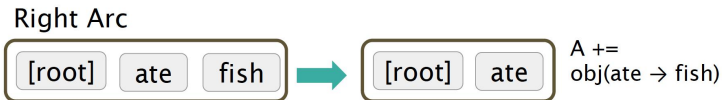
We make a LA_{nsubj} : Left Arc

- the neighbor of the head of the **stack** is a dependent of the head of the **stack**
- add a corresponding link to the **dependency graph** with label *nsubj*
- remove the neighbor from the **stack**

Now we must SHIFT *fish* to the stack so we can empty the buffer



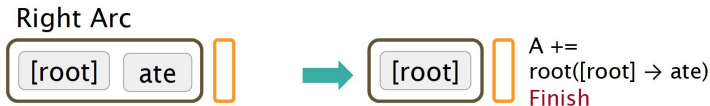
Now the head of the stack, *fish*, depends on *ate* (**dobj**)



We make a RA_{dobj} : Right Arc

- the head of the *stack* is a dependent of its neighbor
- add a corresponding link to the *dependency graph* with label *r*
- remove the head of the *stack*

Like before, the head of the stack, *ate*, depends on *root*, because it is indeed the root of the tree

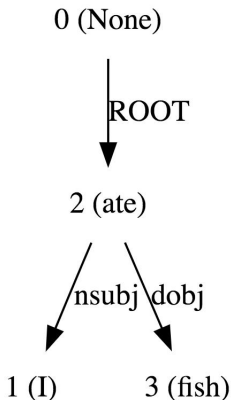


We make another RA_{root} : Right Arc

- the head of the *stack* is a dependent of its neighbor
- add a corresponding link to the *dependency graph* with label *r*
- remove the head of the *stack*

And we're done: the buffer is empty and the graph is full!

This is what our dependency graph looks like:



This is great, but don't you think we left something out?

How do we decide which action to take at each step?

- Use a discriminative classifier (SVM, perceptron, maxent) over all legal moves
 - 3 untyped choices
 - $2|R| + 1$ if we have $|R|$ possible dependency labels ($|R| \approx 40$)
- Features: word at the top of the stack word and its PoS; word at the top of the buffer and its PoS
 - Common approach in the 2000s
 - Sparse feature vectors (up to 10^7 dimensions!)
 - Incomplete, because you can't account for everything (language is very complex!)
 - Feature computation was extremely time-consuming
- Training: use treebanks!
- Fast greedy approach: pick the best option at each step
- This is the basic idea in MaltParser¹

¹J. Nivre and J. Hall, MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. Fourth Workshop on Treebanks and Linguistic Theories, 2005

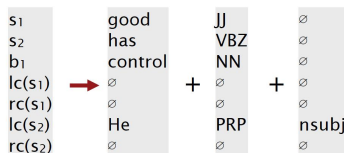
This example was featured in a seminal research paper²

Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	\emptyset
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC (nsubj)	[ROOT has]	[good control .]	$A \cup \text{nsubj}(\text{has}, \text{He})$
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC (amod)	[ROOT has control]	[.]	$A \cup \text{amod}(\text{control}, \text{good})$
RIGHT-ARC (doobj)	[ROOT has]	[.]	$A \cup \text{doobj}(\text{has}, \text{control})$
...
RIGHT-ARC (root)	[ROOT]	[]	$A \cup \text{root}(\text{ROOT}, \text{has})$

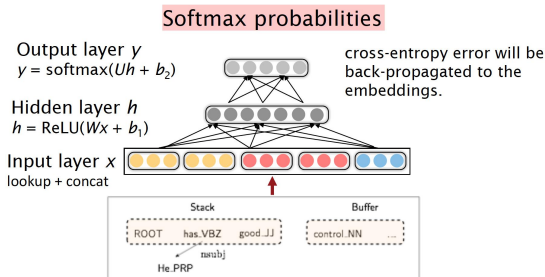
Transition-based dependency parsing is essentially about classifying each transition correctly as a SHIFT, LA_r , or RA_r .

²D. Chen and C. Manning, A Fast and Accurate Dependency Parser using Neural Networks, EMNLP '14

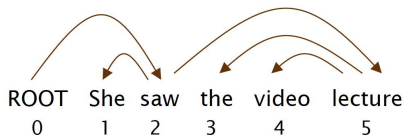
- Represent words as d -dimensional word embeddings with $d \approx 1000$
- Also represent PoS tags and the already known dependency labels as d -dimensional vectors
 - NNS (plural noun) should be close to NN (singular noun)
 - amod (adjective modifier) should be close to advmod (adverbial modifier)
- Extract a set of tokens based on stack/buffer positions
 - For example, head of stack, its neighbor, head of buffer, its neighbor
- Look up their embedding vectors and concatenate them with the PoS and label embedding vectors



- Input layer: concatenated embedding vectors
- Feedforward neural network with ReLu non-linearity
- Softmax gives a probability distribution
- Use cross-entropy loss to measure the error



Google's Syntaxnet essentially uses these same principles



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

$$\text{UAS} = 4 / 5 = 80\%$$

$$\text{LAS} = 2 / 5 = 40\%$$

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Source: Christopher Manning

Research assignment:

- Can you modify the parameter $n_classes$? Why or why not? What does it represent?
- Modify some of the parameters of the model (e.g., *hidden_size*) and see how the performance changes. Explain, in broad terms, the effect of your changes.
- Optional: perform an ablation study where you remove certain features of the model (for example, dropout) and see how the performance changes.

Please answer (some of) these questions in a one-page written report.