Syntax and Parsing

PCL-II
May 18, 2016
Mark Fishel, Uni. Tartu, Estonia

Structure in language

- Sentences can be grouped into bigger sentences, e.g.
  - *S and S*
  - *S but S*
  - *S but S, when S, even though S*
- Phrases also consist of words and phrases
  - **big** table
  - **big and walrus-like** table
  - **not so big, but nevertheless quite impressive** table

Syntax

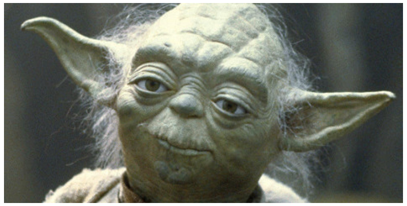
- The study of sentence structure
- Why should we be concerned with it?
  - sentence structure prediction
  - structure-based NLP applications
    - summarization
    - machine translation
    - ...

Syntax for statistical machine translation: pre-reordering

- Input: Ich werde Ihnen die entsprechenden Anmerkungen aushändigen, damit Sie das eventuell bei der Abstimmung übernehmen können.
- Gloss: I will to you the corresponding comments pass on, so that you them perhaps in the vote adopt can.

Syntax for statistical machine translation: pre-reordering

- Input: Ich werde Ihnen die entsprechenden Anmerkungen aushändigen, damit Sie das eventuell bei der Abstimmung übernehmen können.
- Gloss: I will to you the corresponding comments pass on, so that you them perhaps in the vote adopt can.



Syntax for statistical machine translation: pre-reordering

- **Reordered** input: Ich werde aushändigen Ihnen die entsprechenden Anmerkungen, damit Sie können übernehmen das eventuell bei der Abstimmung.
- Gloss: I will pass on to you the corresponding comments, so that you can adopt them perhaps in the vote.

Language properties

- A sentence has no limited length:
  - Mark gave a lecture
  - The students realized that Mark gave a lecture
  - Despite all Mark's efforts the students realized that Mark gave a lecture
  - According to the local news, despite all Mark's efforts the students realized that Mark gave a lecture
  - Laura was happy to announce that according to the local news, despite all Mark's efforts the students realized that Mark gave a lecture
- Potentially an infinite set of sentences
- that we want to handle using a finite model

Language properties

- Ambiguity:
  - The viking killed the pirate with a sword

Language properties

- Ambiguity:
  - (The viking) killed (the pirate) (with a sword)
    vs
  - (The viking) killed (the pirate (with a sword))

Language properties

- Can we ignore structure and handle it on the word sequence level?

Language properties

- Can we ignore structure and handle it on the word sequence level?
    - e.g. “The girl with the flowers is cute”

Language properties

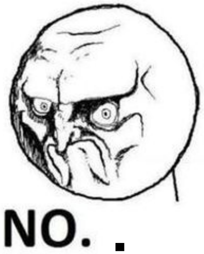
- Can we ignore structure and handle it on the word sequence level?
  - e.g. “The girl with **the flowers is** cute”
  - A 3-gram-based spellchecker will likely suggest: “did you mean ‘**the flowers are**’ ”

Language properties

- Can we ignore structure and handle it on the word sequence level?
  - e.g. “The girl with **the flowers is** cute”
  - A 3-gram-based spellchecker will likely suggest: “did you mean ‘**the flowers are**’ ”
  - e.g. translating “**Legen** sie noch etwas **zu**”:
    - Legen = lay? lie? put?...
    - Zulegen = add

Language properties

- Can we ignore structure and handle it on the word sequence level?

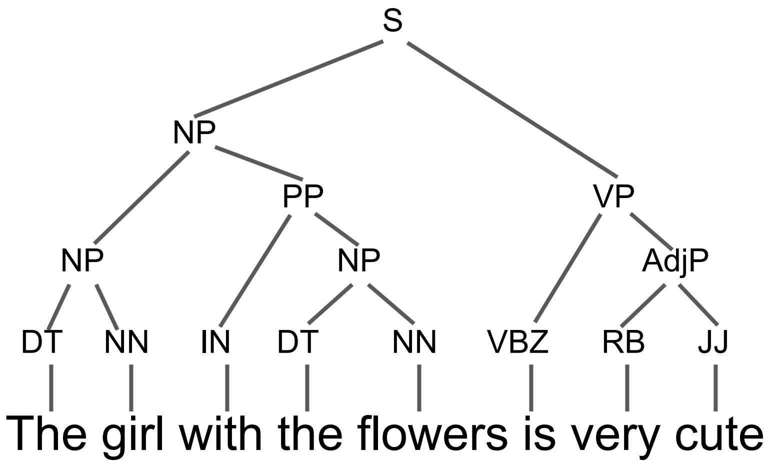


  - e.g. “The girl with **the flowers is** cute”
  - A 3-gram-based spellchecker will likely suggest: “did you mean ‘**the flowers are**’ ”
  - e.g. translating “**Legen** sie noch etwas **zu**”:
    - Legen = lay? lie? put?...
    - Zulegen = add

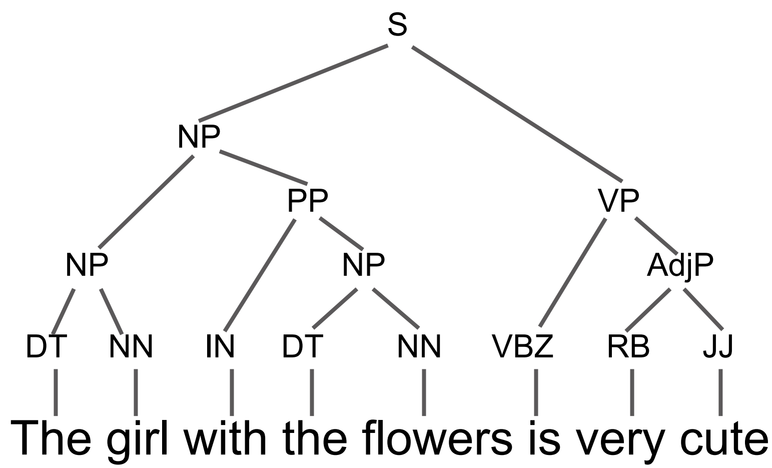
Constituency vs. dependency

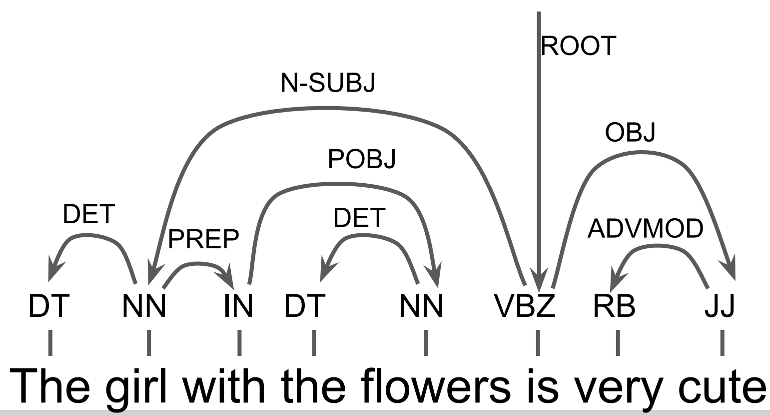
- Constituency structure:
  - sentence-phrase-word hierarchy
  - each phrase labelled with its type
- Dependency structure:
  - every word has a “parent” / head-word
  - every word-parent relation labelled with a type

Constituency vs. dependency

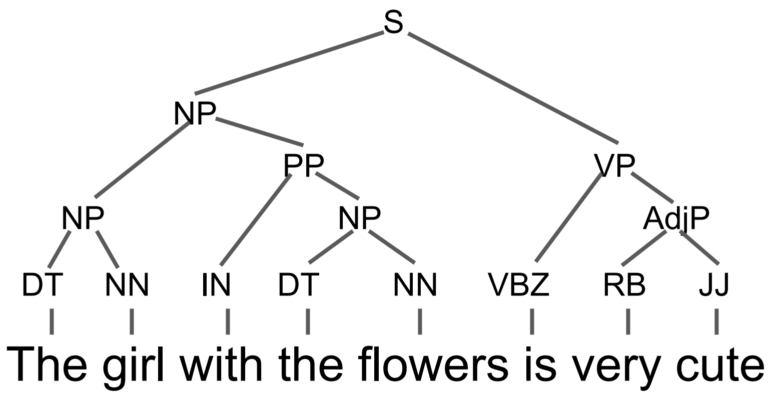


Constituency vs. dependency

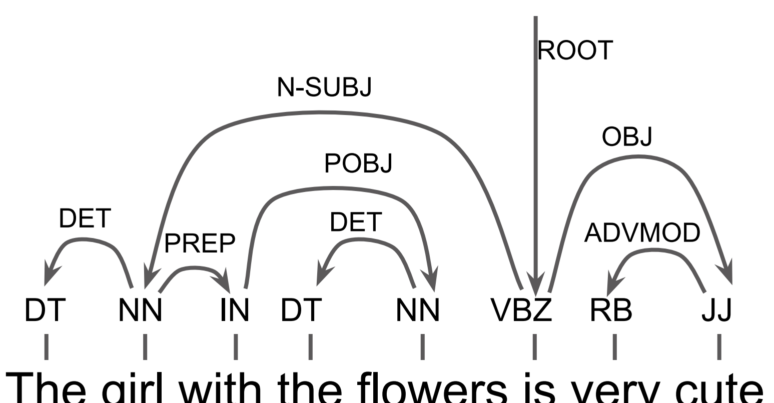




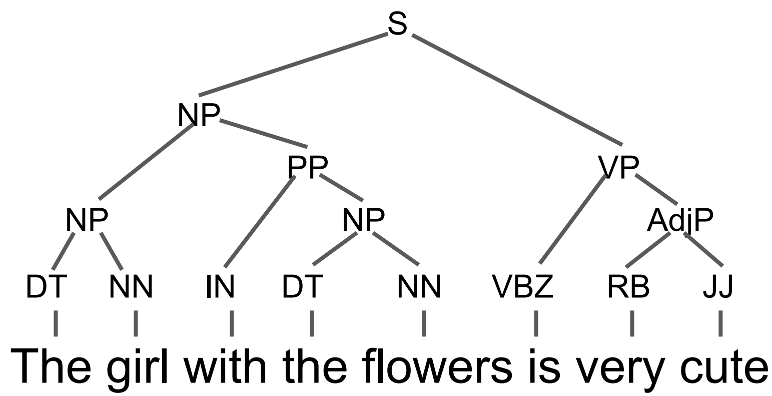
Constituency vs. dependency



$(((\text{The}_{DT}\ \text{girl}_{NN})_{NP}\ (\text{with}_{IN}\ (\text{the}_{DT}\ \text{flowers}_{NN})_{NP})_{PP})_{NP}\ (\text{is}_{VBZ}\ (\text{very}_{RB}\ \text{cute}_{JJ})_{AdjP})_{VP})_{S}$

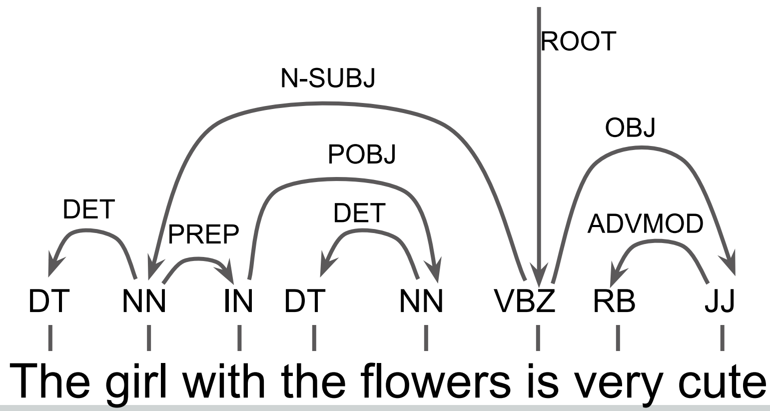


Constituency vs. dependency

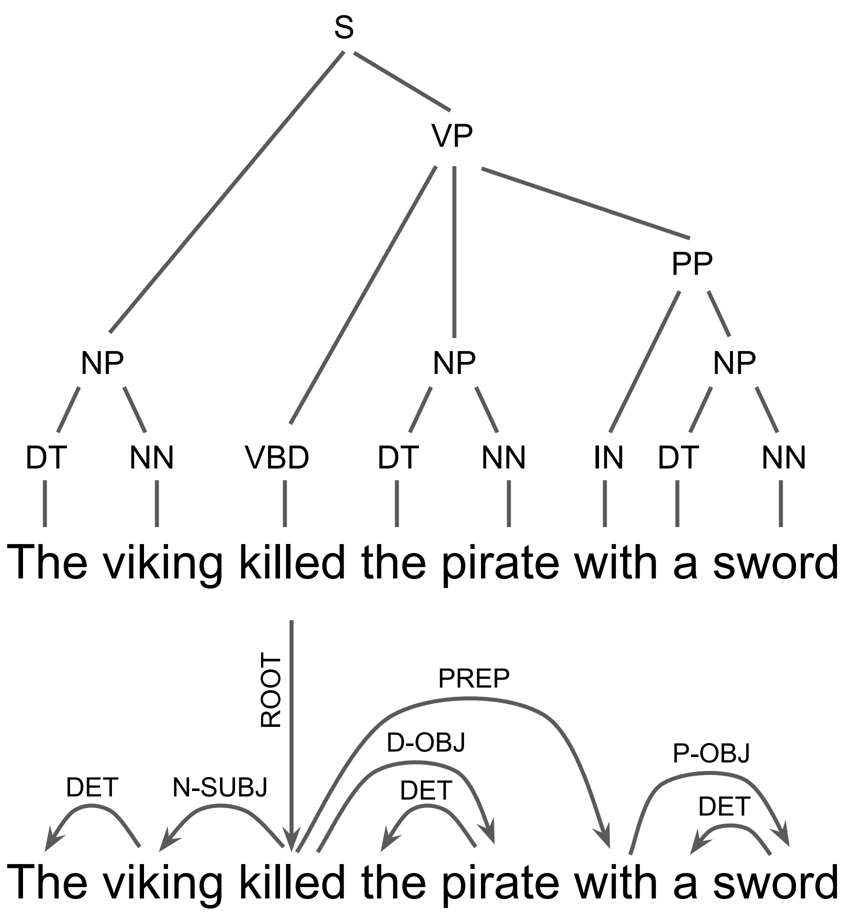


(((The $_{DT}$ girl $_{NN}$) $_{NP}$ (with $_{IN}$ (the $_{DT}$ flowers $_{NN}$) $_{NP}$) $_{PP}$) $_{NP}$ (is $_{VBZ}$ (very $_{RB}$ cute $_{JJ}$) $_{AdjP}$) $_{VP}$) $_{S}$

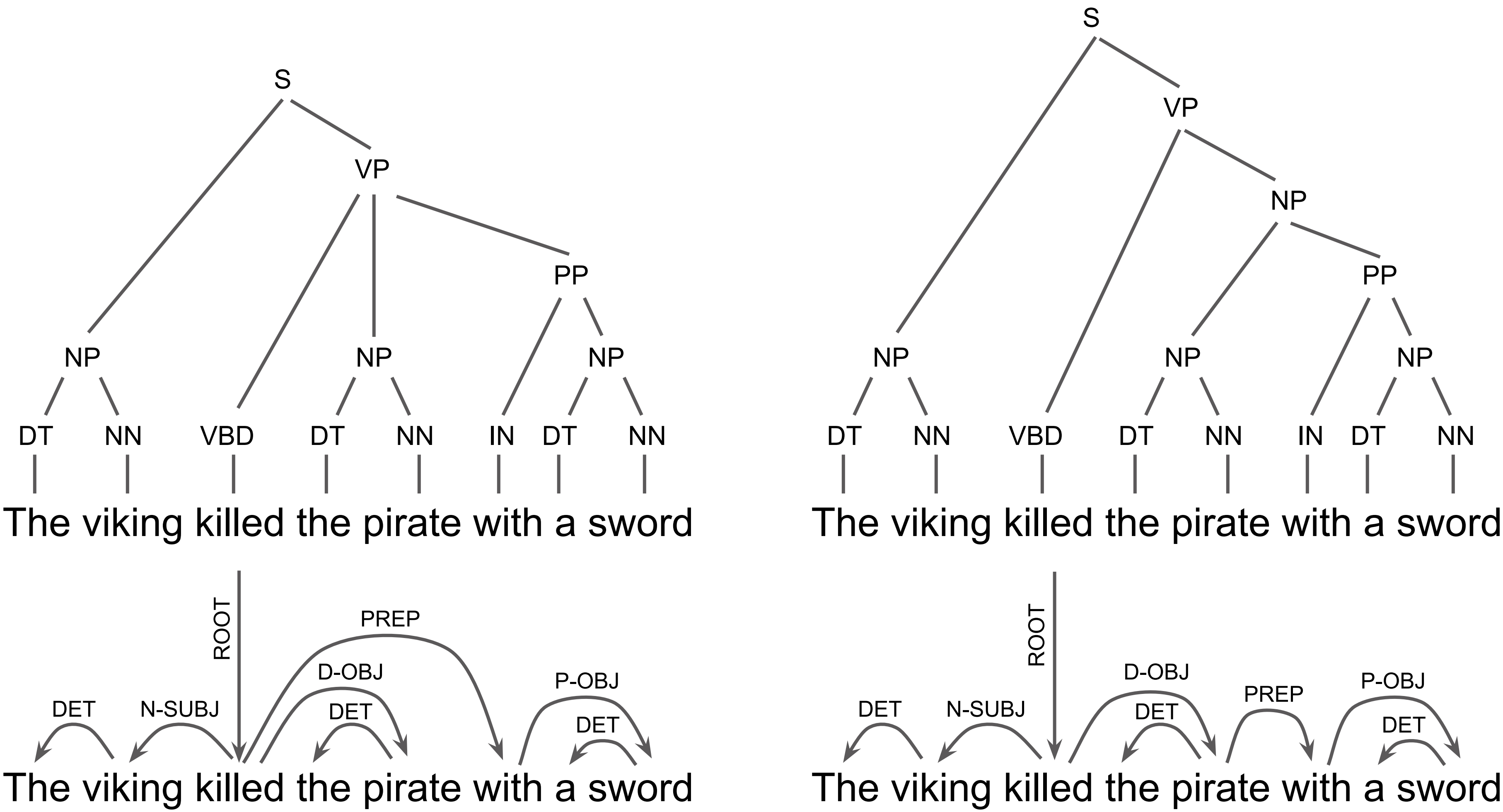
(((The$_{DET}$ girl$_{NN}$) (with$_{IN}$ (the$_{DET}$ flowers$_{NN}$)$_{POBJ}$)$_{PREP}$)$_{N\text{-}SUBJ}$ is$_{VBZ}$ (very$_{ADVMOD}$ cute$_{JJ}$)$_{OBJ}$)$_{ROOT}$



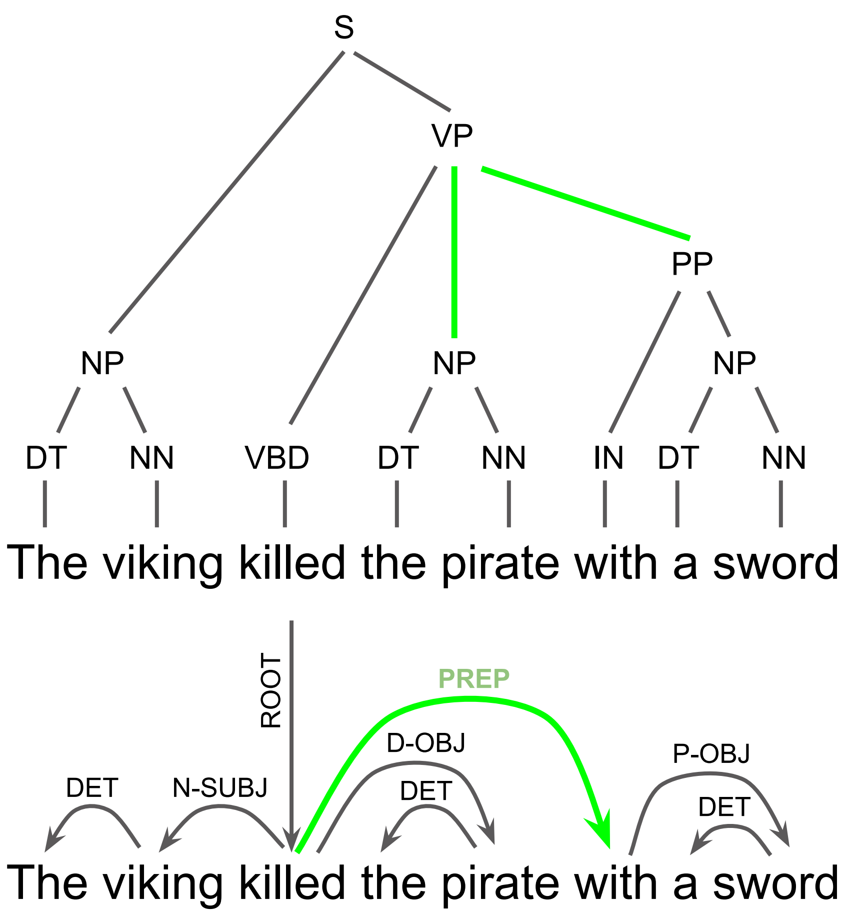
Constituency vs. dependency

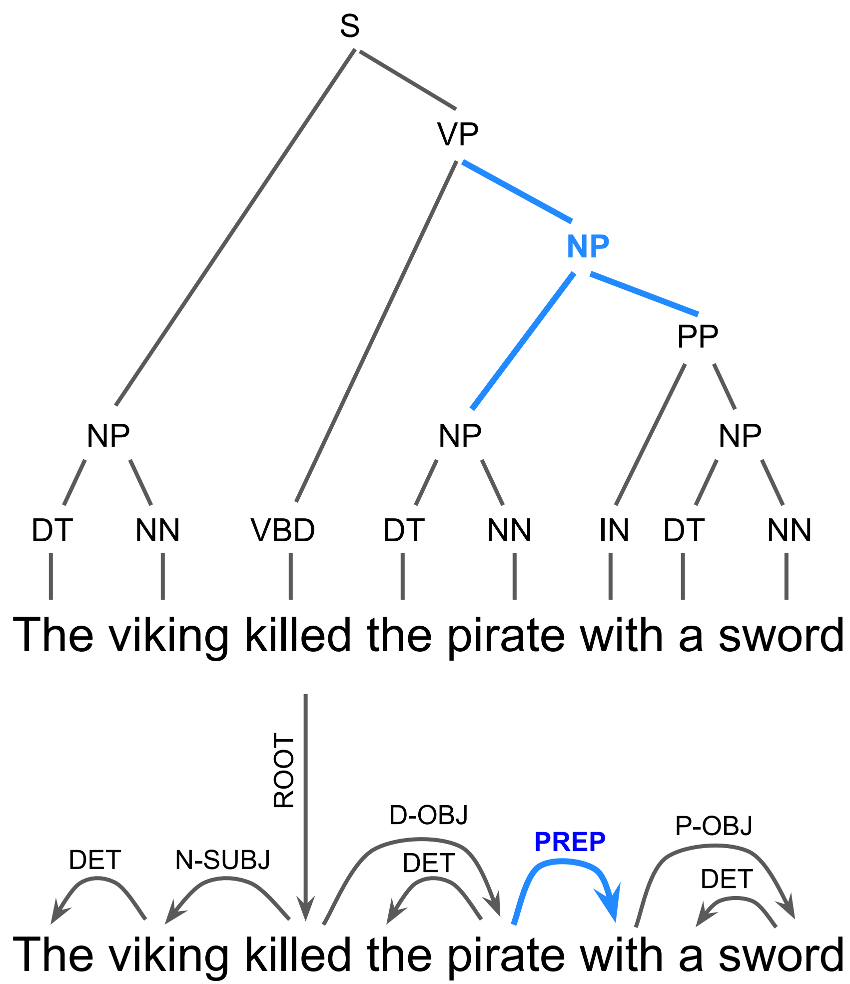


Constituency vs. dependency

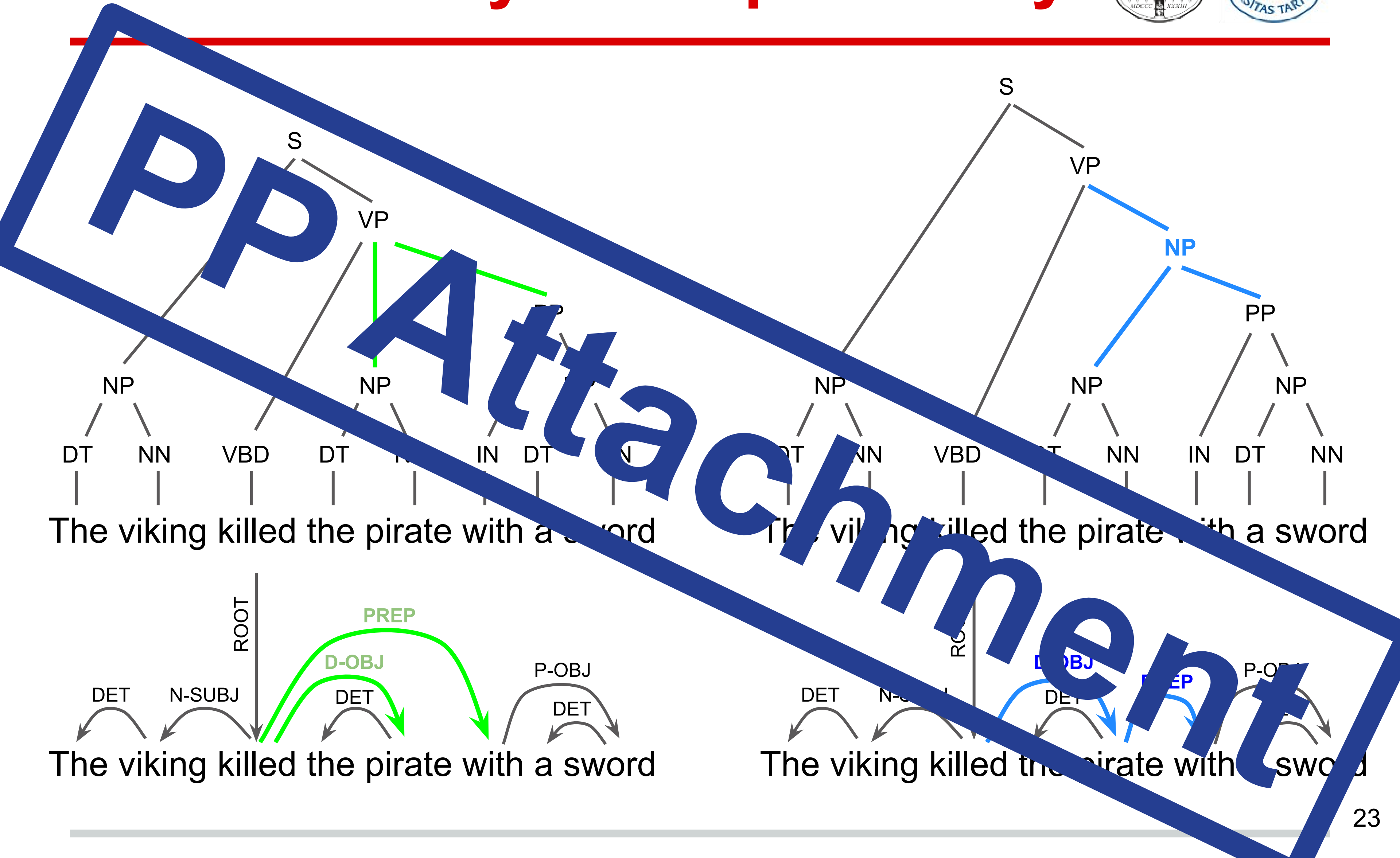


Constituency vs. dependency





Constituency vs. dependency



Next

- Grammars
  - describing structures
- CYK parsing
  - analyzing sentence structure
- Probabilistic grammars
  - structure + probabilities
- Grammar acquisition
  - from treebanks to grammars, automatically

Grammar

- A set of rules describing which sentences are acceptable for a particular language
- Context-free grammars (CFG): classical approach to consistency syntax
    - even though dependency syntax can also be described with the same kind of grammar
- Key point: describe the permitted structures one sub-tree at a time

Context-free grammar

Example:

- S → NP VP

- NP → N
- NP → JJ NP

- VP → V NP
- VP → V

- N → dogs
- N → cats
- N → stuff

- JJ → big
- JJ → black

- V → chase
- V → eat
- V → sleep

Context-free grammar

Example:

- S → NP VP

- NP → N
- NP → JJ NP

- VP → V NP
- VP → V

- N → dogs
- N → cats
- N → stuff

- JJ → big
- JJ → black

- V → chase
- V → eat
- V → sleep

Sentences:

- dogs chase cats
- cats eat
- big cats eat big dogs
- big black dogs sleep

Context-free grammar

Example:

- S → NP VP

- NP → N
- NP → JJ NP

- VP → V NP
- VP → V

- N → dogs
- N → cats
- N → stuff

- JJ → big
- JJ → black

- V → chase
- V → eat
- V → sleep

Sentences:

- dogs chase cats
- cats eat
- big cats eat big dogs
- big black dogs sleep

But also

- black dogs sleep big cats
- black black big black dogs eat

Context-free grammar

Example:

- S → NP VP
- NP → N
- NP → JJ NP
- VP → V NP
- VP → V
- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep

Sentences:

- dogs chase cats
- cats eat
- big cats eat big dogs
- big black dogs sleep

But also

- black dogs sleep big cats
- black black big black dogs eat

Grammar:

- a set of terminal symbols (dogs, stuff, big, ...)
- a set of non-terminal symbols (N, JJ, NP, ...)
- a set of rules to turn non-terminals into terminals and other non-terminals
- a (set of) starting symbol(s)

Grammatical sentences

- **Sentence form** of a grammar:
  - S is a sentence form
  - $\alpha\gamma\beta$ is a sentence form if
    - there is a sentence form $\alpha B\beta$
    - there is a rule $B \rightarrow \gamma$ within the grammar
- **Grammatical sentence** = a sentence form with only terminal symbols in it

Parsing

Task: given a grammar and a sentence:

- is the sentence part of the given grammar?
- what is its structure/rule sequence that would generate it starting from S?

Parsing

- Is the sentence parseable with the given grammar, a naive solution:
  - generate all trees that the grammar can generate
  - check if given sentence is among them
- Why naive?

Parsing

- Is the sentence parseable with the given grammar, a naive solution:
  - generate all trees that the grammar can generate
  - check if given sentence is among them
- Why naive?
  - the set of sentences that the grammar accepts is in general exponential
  - due to recursion it can also be infinite

Chart parsing

- A solution: dynamic programming
- Find partial trees only once, store in a chart and reuse
- Cocke-Younger-Kasami (CYK) algorithm:
  - also called CKY algorithm
  - bottom-up parsing
  - list of chart entries per word subsequence

CYK parsing

- Requirement: the grammar must be in the "Chomsky normal form" (CNF), which allows only 2 kinds of rules:
  - $A \rightarrow a$ (a non-terminal into a terminal)
  - $A \rightarrow BC$ (a non-terminal into 2 non-terminals)

CYK parsing

- Requirement: the grammar must be in the "Chomsky normal form" (CNF), which allows only 2 kinds of rules:
  - A → a (a non-terminal into a terminal)
  - A → BC (a non-terminal into 2 non-terminals)
- Any grammar can be put into CNF, e.g.
  - **original rule:** **CNF rules:**

original rule	CNF rules
$VP \rightarrow V\ NP\ PP$	$VP \rightarrow V\ VP_1$ $VP_1 \rightarrow NP\ PP$
$NP \rightarrow little\ N$	$NP \rightarrow ADJ_1\ N$ $ADJ_1 \rightarrow little$

CYK parsing

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	big	dogs	chase	black	cats

- S → NP VP
- NP → N
- NP → JJ NP
- VP → VP NP
- VP → V

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep

- A cell for every subsequence (total num: len^2)

CYK parsing

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	big	dogs	chase	black	cats

- S → NP VP
- ~~NP → N~~
- NP → JJ NP
- VP → VP NP
- ~~VP → V~~

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- A cell for every subsequence (total num: len^2)

CYK parsing

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	big	dogs	chase	black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff

- JJ → big
- JJ → black

- V → chase
- V → eat
- V → sleep

- NP → dogs
- NP → cats
- NP → stuff

- VP → chase
- VP → eat
- VP → sleep

- A cell for every subsequence (total num: len^2)

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	big	dogs	chase	black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	dogs	chase	black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	chase	black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 1

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- Find a non-terminal$_{\text{(PoS-tag)}}$ for each terminal$_{\text{(token)}}$

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	big dogs chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	S big dogs | chase	dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	S big dogs | chase	- dogs chase black	chase black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	big dogs chase black	dogs chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	- big dogs chase black	dogs chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	- big dogs chase black	S dogs | chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

5	big dogs chase black cats				
4	- big dogs chase black	S dogs | chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: step 2

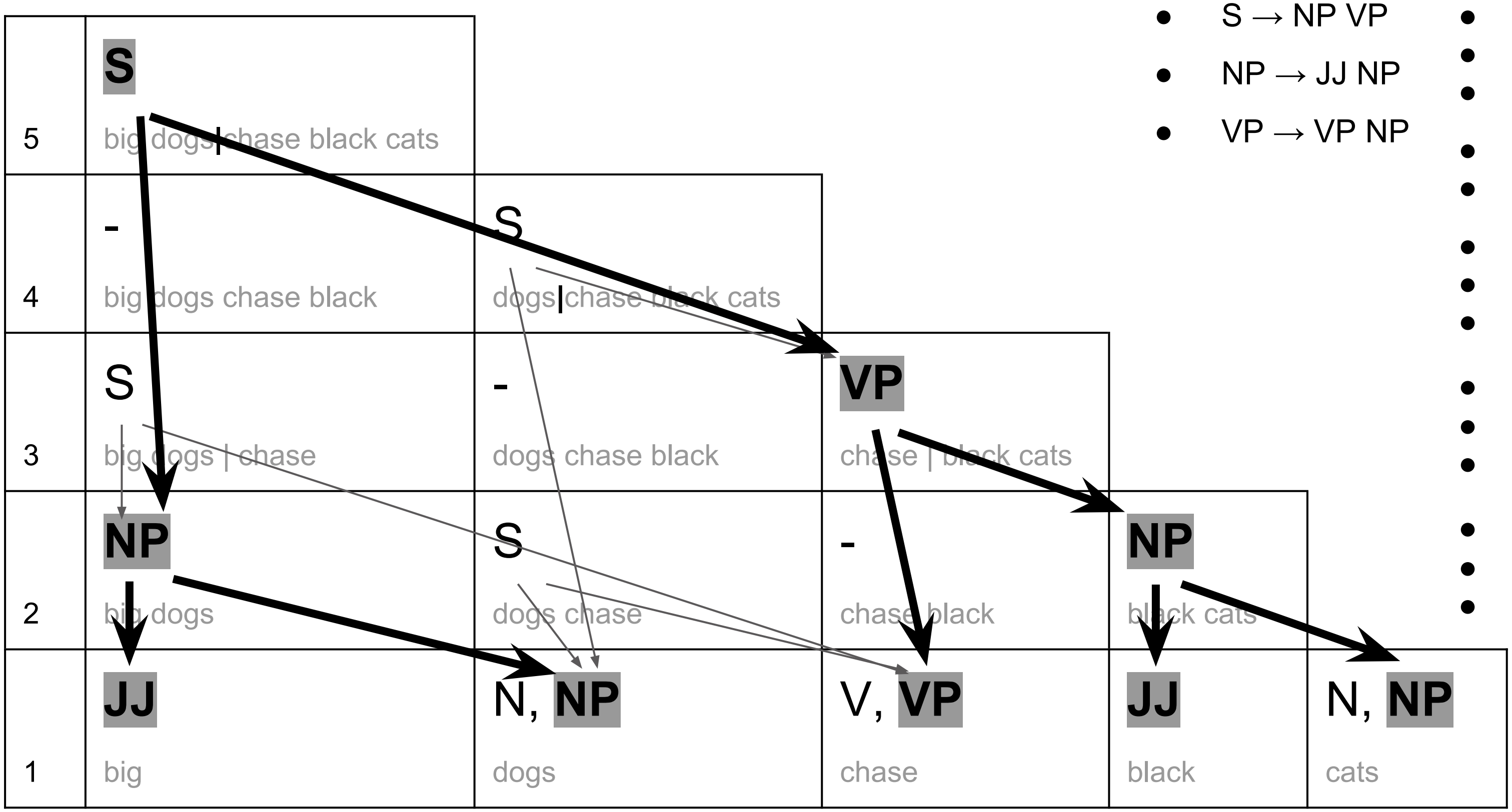
5	S big dogs | chase black cats				
4	- big dogs chase black	S dogs | chase black cats			
3	S big dogs | chase	- dogs chase black	VP chase | black cats		
2	NP big dogs	S dogs chase	- chase black	NP black cats	
1	JJ big	N, NP dogs	V, VP chase	JJ black	N, NP cats

- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- `for i in 2..N:` try splitting each i-word sequence into two sub-sequences (i-1 different ways)
- check if both sub-sequences have at least 1 parse available in the table

CYK parsing: ready!



- S → NP VP
- NP → JJ NP
- VP → VP NP

- N → dogs
- N → cats
- N → stuff
- JJ → big
- JJ → black
- V → chase
- V → eat
- V → sleep
- NP → dogs
- NP → cats
- NP → stuff
- VP → chase
- VP → eat
- VP → sleep

- back-track the tree(s) starting from the top cell

Parsing Pseudocode

```
def parse(grammar, sentence):
    cnfGrammar = getCNF(grammar)

    memory = dict(subseq) of lists of subtrees
        # subtree: (
        #    NonTerminal,
        #   ( LeftSubSequence, LeftNonTerminal),
        #   ( RightSubSequence, RightNonTerminal) )

    for token in sentence:
        memory[token] = list of possible PoS tags for token
            # or PoS-tagging with 1 tag per token
```

Parsing Pseudocode

```
def parse(grammar, sentence):
    …(continued)…

    for sequenceLen in (2, …, len(sentence)):
        for subSequence in (subSequences of sentence with length sequenceLen):
            for splitPoint in (1, …, sequenceLen - 1):
                (leftSubSeq, rightSubSeq) = split subSequence on splitPoint

                for LeftNT in memory[leftSubSeq] NonTerminals:
                    for RightNT in memory[rightSubSeq] NonTerminals:
                        if cnfGrammar has a rule X → LeftNT RightNT:
                            add it to the list under memory[subSequence]

    the result(s) are in memory[sentence]
```

Parsing Pseudocode

```
def parse(grammar, sentence):
    …(continued)…

    for sequenceLen in (2, …, len(sentence)):
        for subSequence in (subSequences of sentence with length sequenceLen):
            for splitPoint in (1, …, sequenceLen - 1):
                (leftSubSeq, rightSubSeq) = split subSequence on splitPoint

                for LeftNT in memory[leftSubSeq] NonTerminals:
                    for RightNT in memory[rightSubSeq] NonTerminals:
                        if cnfGrammar has a rule X → LeftNT RightNT:
                            add it to the list under memory[subSequence]

    the result(s) are in memory[sentence]
```

Complexity?

Parsing Pseudocode

```
def parse(grammar, sentence):
    …(continued)…

    for sequenceLen in (2, …, len(sentence)):
        for subSequence in (subSequences of sentence with length sequenceLen):
            for splitPoint in (1, …, sequenceLen - 1):
                (leftSubSeq, rightSubSeq) = split subSequence on splitPoint

                for LeftNT in memory[leftSubSeq] NonTerminals:
                    for RightNT in memory[rightSubSeq] NonTerminals:
                        if cnfGrammar has a rule X → LeftNT RightNT:
                            add it to the list under memory[subSequence]

    the result(s) are in memory[sentence]
```

Complexity? $O(n^3)$

Parsing in NLTK

```
import nltk

grammar1 = nltk.CFG.fromstring("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw" | "ate" | "walked"
  NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
  Det -> "a" | "an" | "the" | "my"
  N -> "man" | "dog" | "cat" | "telescope" | "park"
  P -> "in" | "on" | "by" | "with"
  """)

sent = "Mary saw Bob".split()
parser = nltk.parse.chart.BottomUpChartParser(grammar1)
for tree in parser.parse(sent):
   print tree
#(S (NP Mary) (VP (V saw) (NP Bob)))
```

Ambiguity

- How to decide between several possible parses?

Ambiguity

- How to decide between several possible parses?
- One solution: probabilistic CFG (PCFG)

Ambiguity

- How to decide between several possible parses?
- One solution: probabilistic CFG (PCFG)

CFG:

- S → NP VP
- NP → N
- NP → JJ NP
- VP → VP NP
- VP → V
- N → dogs
- NP → dogs

...

Ambiguity

- How to decide between several possible parses?
- One solution: probabilistic CFG (PCFG)

CFG:

- S → NP VP
- NP → N
- NP → JJ NP
- VP → VP NP
- VP → V
- N → dogs
- NP → dogs

...

PCFG:

- S → NP VP *(p = 1)*
- NP → N *(p = 0.6)*
- NP → JJ NP *(p = 0.4)*
- VP → VP NP *(p = 0.32)*
- VP → V *(p = 0.68)*
- N → dogs *(p = 0.17)*
- NP → dogs *(p = 0.17)*

...

PCFG Parsing

Algorithm same as CKY, but with a Viterbi twist

- each possible parse 1 step down is kept with its probability
- probabilities are multiplied
- link to the likeliest predecessor is kept
- back-tracking follows the most likeliest predecessor links

PCFG Parsing in NLTK

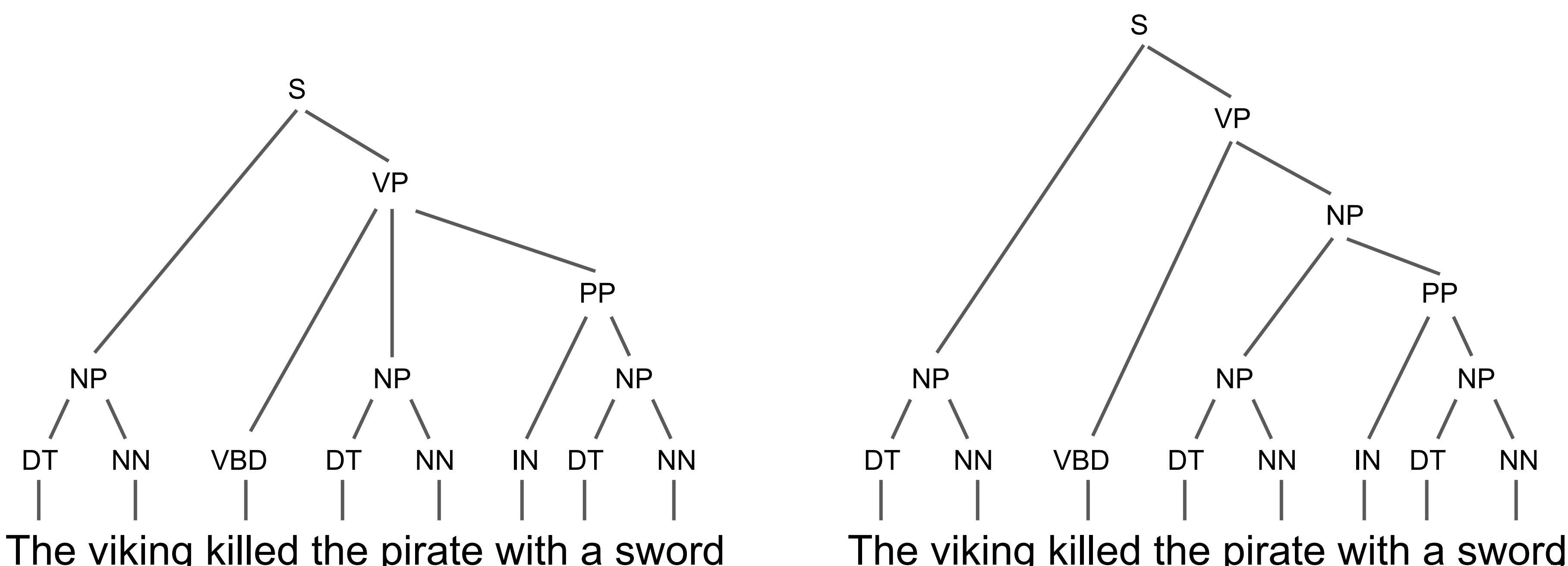
```
import nltk

grammar2 = nltk.PCFG.fromstring("""
  S -> NP VP              [1.0]
  VP -> V NP              [0.217]
  VP -> V NP PP           [0.53]
  ...
  """)

sent = "Mary saw Bob".split()

viterbi_parser = nltk.ViterbiParser(grammar2)
for tree in viterbi_parser.parse(sent):
   print tree
```

Grammar learning

- parsing = having a grammar, find a sentence structure
- Q: where do we get a grammar from?

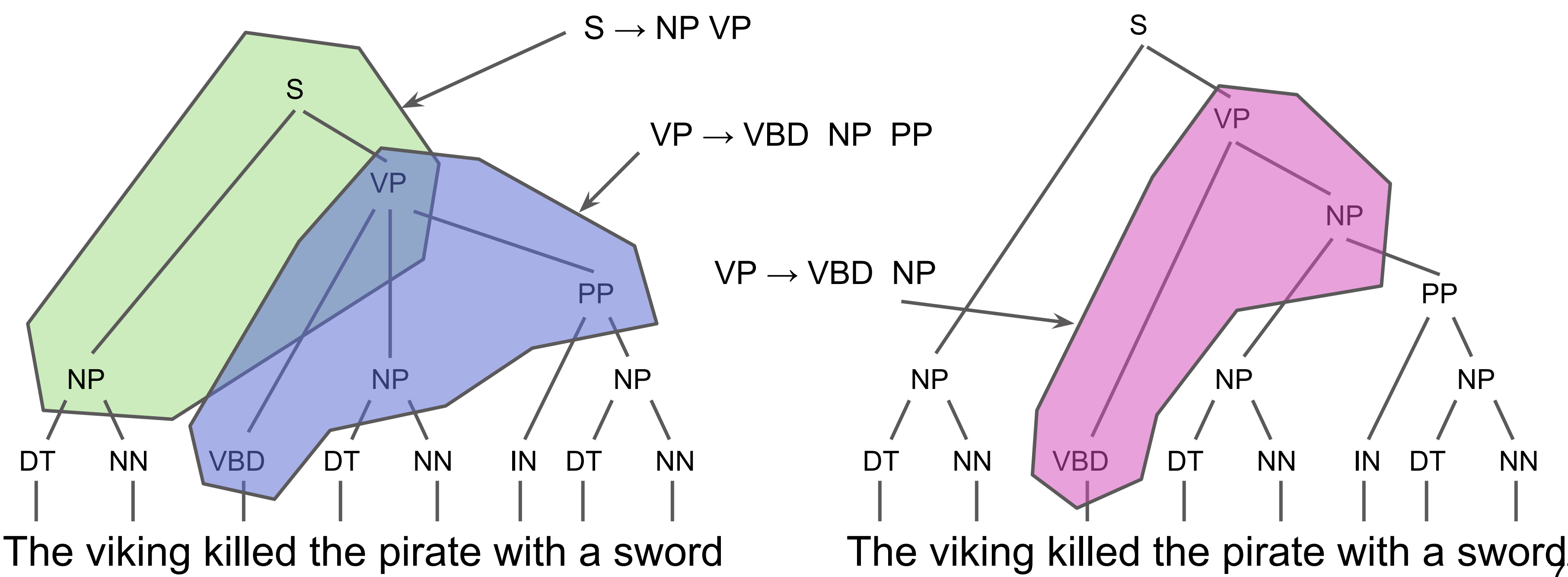
Grammar Learning

- Q: where do we get a grammar from?
- A: we learn it!
  - from a treebank = syntactically annotated corpus



Grammar Learning

- Q: where do we get a grammar from?
- A: we learn it!
  - from a treebank = syntactically annotated corpus
  - subtree = production



Grammar Learning

```
from nltk.corpus import treebank
treebank.sents()[314]
#['The', 'offer', 'is', 'being', 'launched', …]
```

Grammar Learning

```
from nltk.corpus import treebank

treebank.sents()[314]
#['The', 'offer', 'is', 'being', 'launched', …]

treebank.parsed_sents()[314]
#Tree('S', [Tree('NP-SBJ-45', [Tree('DT', ['The']), Tree
('NN', ['offer'])]), Tree('VP', [Tree('VBZ', ['is']), Tree
('VP', [Tree('VBG', ['being']), …)
```

Grammar Learning

```
from nltk.corpus import treebank

treebank.sents()[314]
#['The', 'offer', 'is', 'being', 'launched', …]

treebank.parsed_sents()[314]
#Tree('S', [Tree('NP-SBJ-45', [Tree('DT', ['The']), Tree
('NN', ['offer'])]), Tree('VP', [Tree('VBZ', ['is']), Tree
('VP', [Tree('VBG', ['being']), …)

treebank.parsed_sents()[314].productions()[4]
#VP -> VBZ VP
```

Grammar Learning

```
from nltk.corpus import treebank

treebank.sents()[314]
#['The', 'offer', 'is', 'being', 'launched', …]

treebank.parsed_sents()[314]
#Tree('S', [Tree('NP-SBJ-45', [Tree('DT', ['The']), Tree
('NN', ['offer'])]), Tree('VP', [Tree('VBZ', ['is']), Tree
('VP', [Tree('VBG', ['being']), …)

treebank.parsed_sents()[314].productions()[4]
#VP -> VBZ VP

treebank.parsed_sents()[314].productions()[0].lhs()
#VP

treebank.parsed_sents()[0].productions()[0].rhs()
#(VBZ, VP)
```

Grammar Learning

```
from nltk.corpus import treebank
from nltk.grammar import CFG, Nonterminal

tbank_productions = set(production for sent in
        treebank.parsed_sents() for production in
        sent.productions())

tbank_grammar = CFG(Nonterminal('S'),
        list(tbank_productions))
```

Weighed Grammar Learning

Learning the weights/probabilities:

- $p(A \rightarrow \gamma) = p(\gamma \mid A)$
  (must sum to 1 over all γ for any A)

- $p(A \rightarrow \gamma) = \mathrm{count}(A \rightarrow \gamma) \,/\, \mathrm{count}(A)$

(γ: any sequence of terminals/non-terminals)

Weighed Grammar Learning

```
import nltk
from nltk.corpus import treebank
from nltk.grammar import Nonterminal

tbank_production_list = [production for sent in
       treebank.parsed_sents() for production in
       sent.productions()]

tbank_prob_grammar = nltk.induce_pcfg(Nonterminal('S'),
       tbank_production_list)
```

To play with:

- English: http://nlp.stanford.edu:8080/parser/
- German: http://kitt.cl.uzh.ch/kitt/parzu/

To play with:

- Train your own dependency parser: http://www.maltparser.org/
- Use MaltParser through NLTK: `nltk.parse.malt.MaltParser` (http://www.nltk.org/api/nltk.parse.html)

Further reading:

- NLTK book
  - ch. 8 “Analyzing Sentence Structure”,
    ch. 9 “Building Feature Based Grammars”
- FSNLP, Manning&Schütze:
  - ch. 11 “Probabilistic CFGs”,
    ch. 12 “Probabilistic parsing”
- Speech & Language Processing, Jurafsky & Martin (2nd edition):
  - part III: Syntax

Procrastinators, unite!

...tomorrow