Natural Language Processing

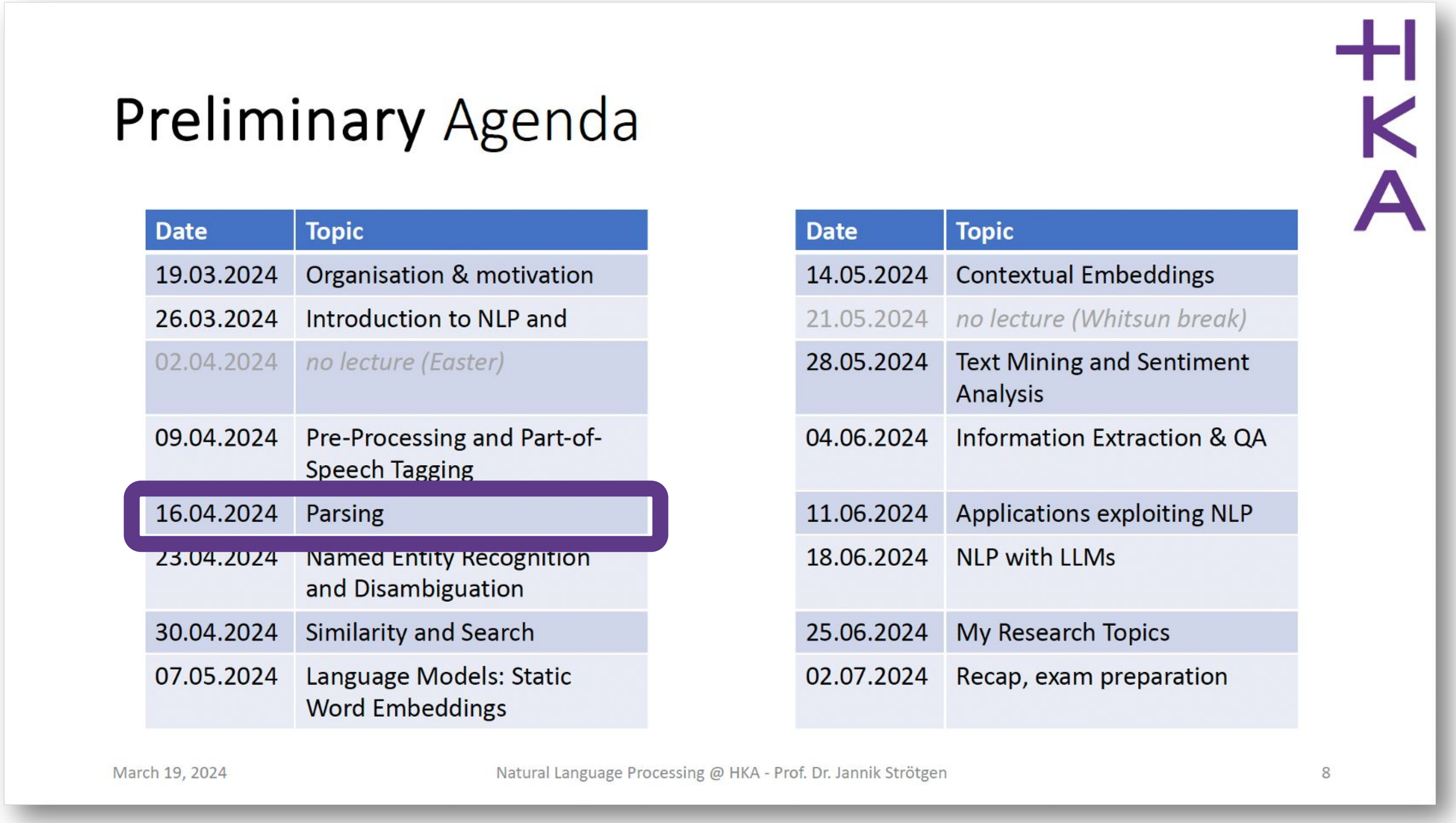
Prof. Dr. Jannik Strötgen
jannik.stroetgen@h-ka.de

Summer 2024

Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

Preliminary Agenda



Preliminary Agenda

Date	Topic
19.03.2024	Organisation & motivation
26.03.2024	Introduction to NLP and
02.04.2024	*no lecture (Easter)*
09.04.2024	Pre-Processing and Part-of-Speech Tagging
16.04.2024	Parsing
23.04.2024	Named Entity Recognition and Disambiguation
30.04.2024	Similarity and Search
07.05.2024	Language Models: Static Word Embeddings

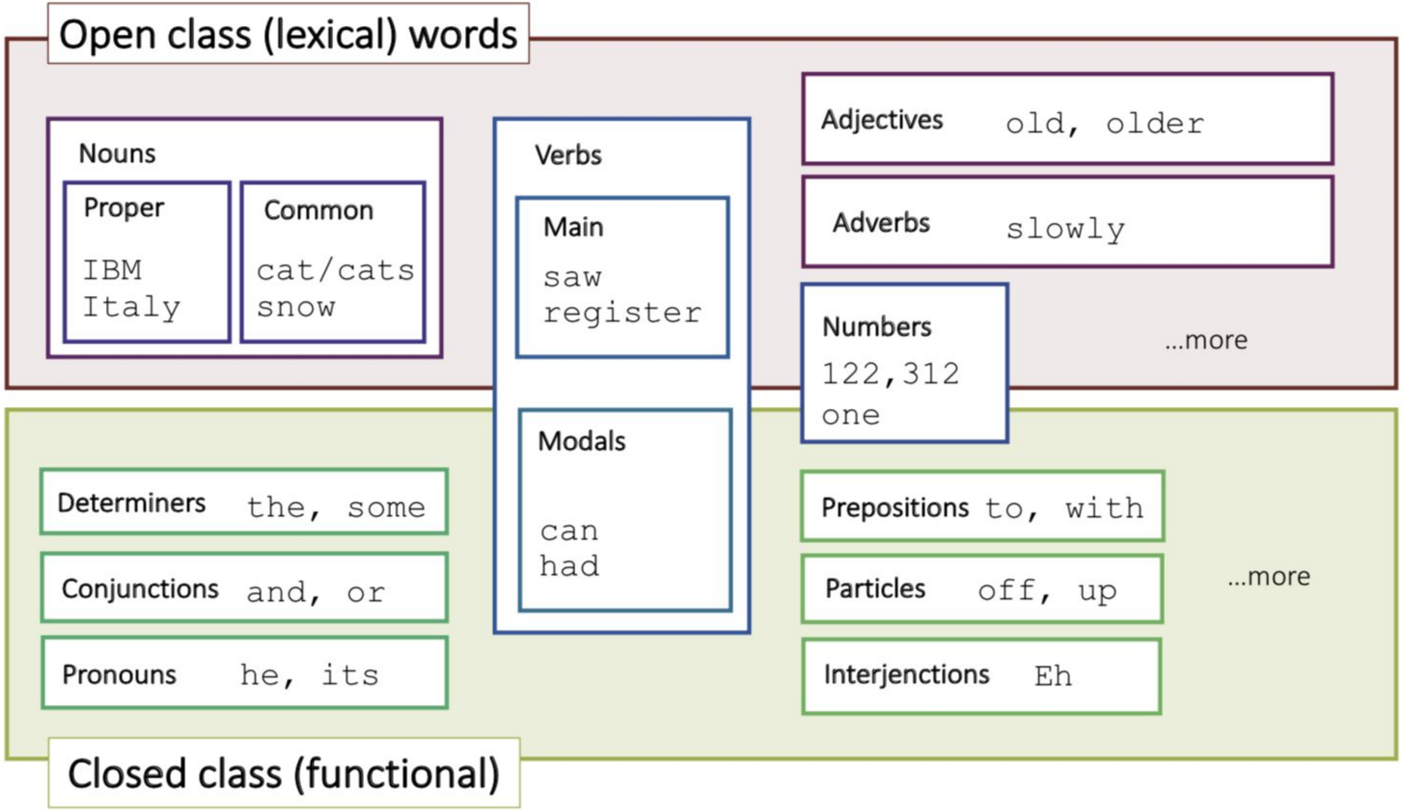
Date	Topic
14.05.2024	Contextual Embeddings
21.05.2024	*no lecture (Whitsun break)*
28.05.2024	Text Mining and Sentiment Analysis
04.06.2024	Information Extraction & QA
11.06.2024	Applications exploiting NLP
18.06.2024	NLP with LLMs
25.06.2024	My Research Topics
02.07.2024	Recap, exam preparation

March 19, 2024 Natural Language Processing @ HKA - Prof. Dr. Jannik Strötgen 8

Recap

Part of Speech Tagging

Parts of Speech



Open class (lexical) words

Nouns

Proper
`IBM`
`Italy`

Common
`cat/cats`
`snow`

Verbs

Main
`saw`
`register`

Modals
`can`
`had`

Adjectives `old, older`

Adverbs `slowly`

Numbers
`122,312`
`one`

...more

Determiners `the, some`

Conjunctions `and, or`

Pronouns `he, its`

Prepositions `to, with`

Particles `off, up`

Interjections `Eh`

...more

Closed class (functional)

POS Tagging

Part of speech tagging:

The process of associating POS labels with each token in a text.

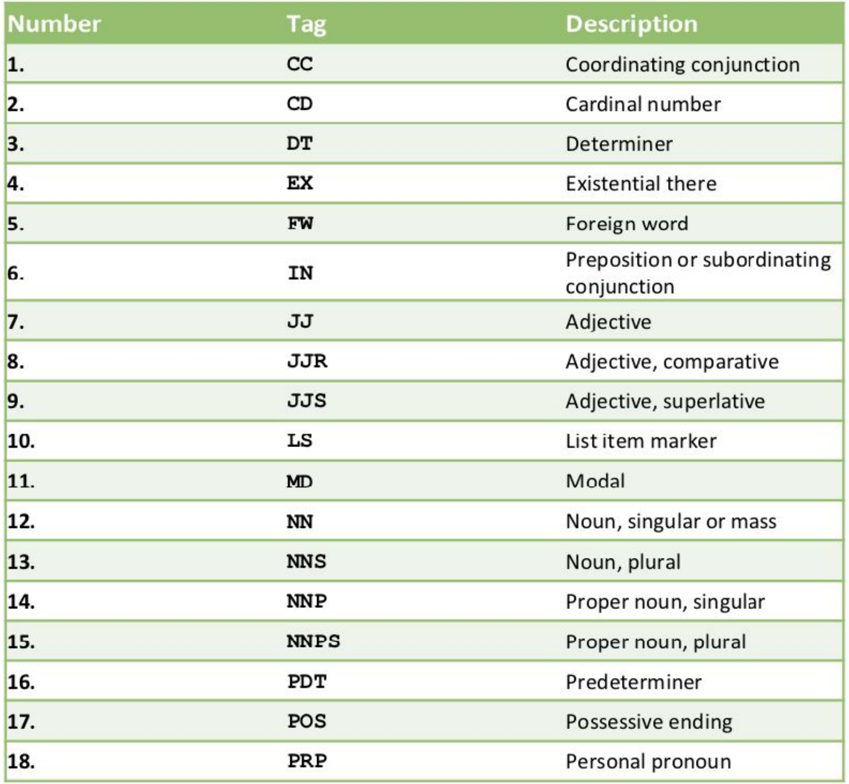
Probabilistic methods

- Use a **corpus for training**
- Obtain POS probabilities from **manual annotations** of the corpus

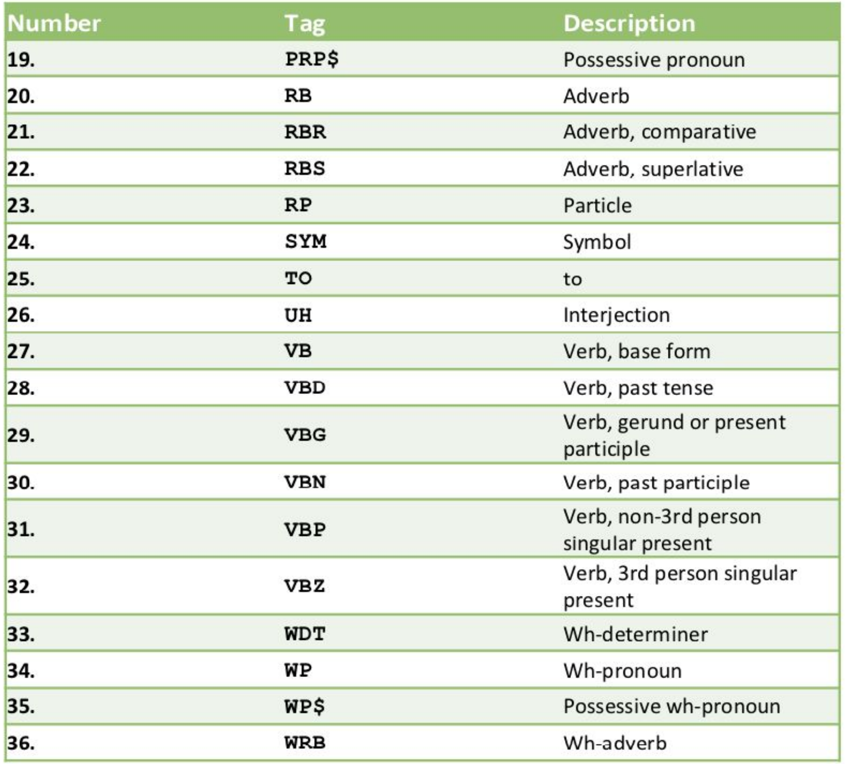
Rule-based methods

- Use **linguistic knowledge** to solve ambiguous cases
- Knowledge is mapped to rules

POS Tagsets (Example: Penn Treebank)



Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun



Number	Tag	Description
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Outline

(Preliminary) Outline of the Course

1. Foundations and Pre-processing
2. Part-of-speech Tagging
3. **Parsing**
4. Named Entity Recognition and Linking
5. Similarity and Search
6. Language Models: Static Word Embeddings
7. Contextual Language Models
8. Text Mining (Classification, Clustering, and Topic Models)
9. Opinion Mining and Sentiment Analysis
10. Relation Extraction and Question Answering
11. Applications in Document Analysis

[3] Parsing

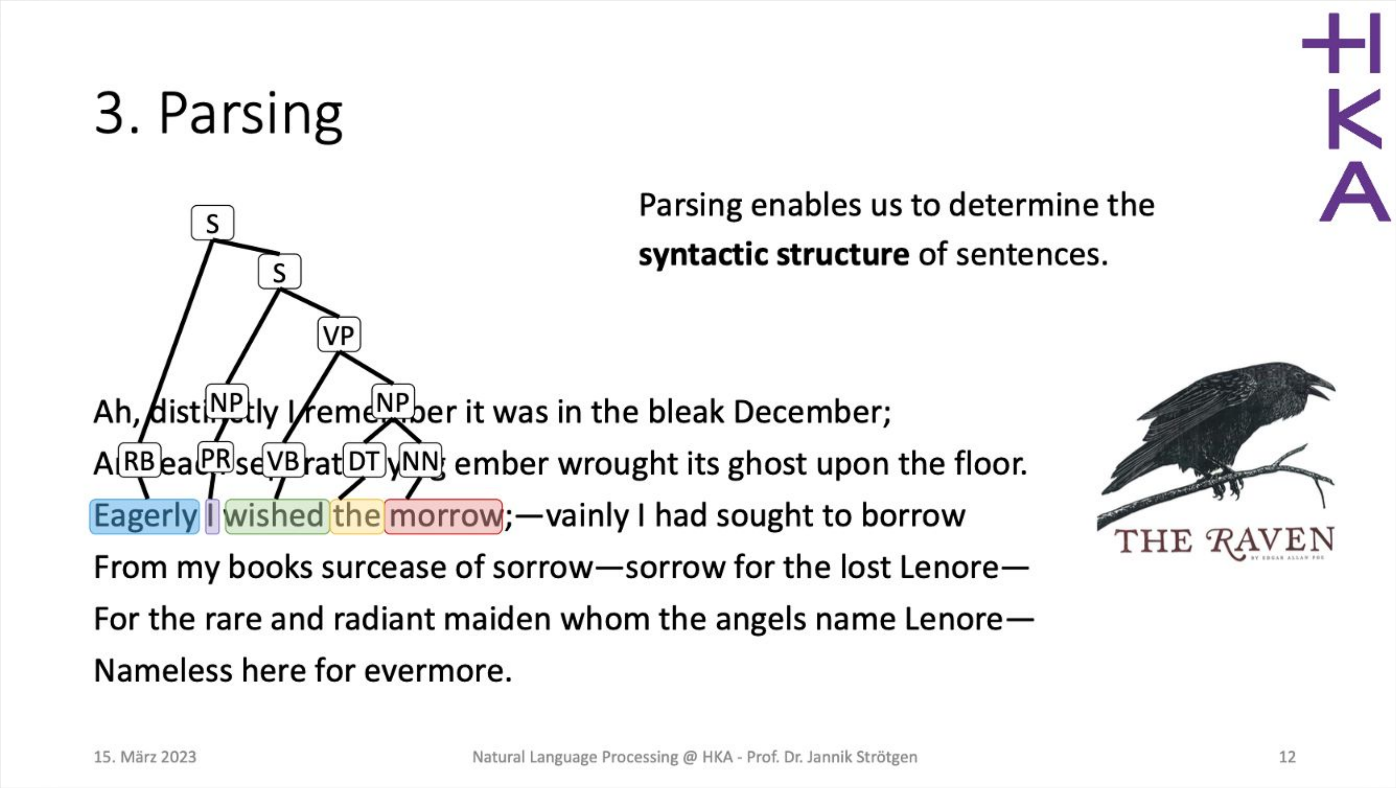
Outline for Today

Parsing

1. Motivation: What is Parsing?
2. Formal Grammar
3. Phrase Structure
4. Constituency Parsing: Task and Challenges
5. Constituency Parsing Algorithms
   - Leftmost Derivation
   - Shift-Reduce Parser
   - Cocke-Younger-Kasami Parser
   - Statistical Parsing
   - Lexicalized Parsing
6. Dependency Parsing

Motivation:
What Is Parsing?

From the Introduction



3. Parsing

Parsing enables us to determine the **syntactic structure** of sentences.

Ah, distinctly I remember it was in the bleak December;
And each separate dying ember wrought its ghost upon the floor.
Eagerly I wished the morrow;—vainly I had sought to borrow
From my books surcease of sorrow—sorrow for the lost Lenore—
For the rare and radiant maiden whom the angels name Lenore—
Nameless here for evermore.

15. März 2023 Natural Language Processing @ HKA - Prof. Dr. Jannik Strötgen 12

One (recent) Example

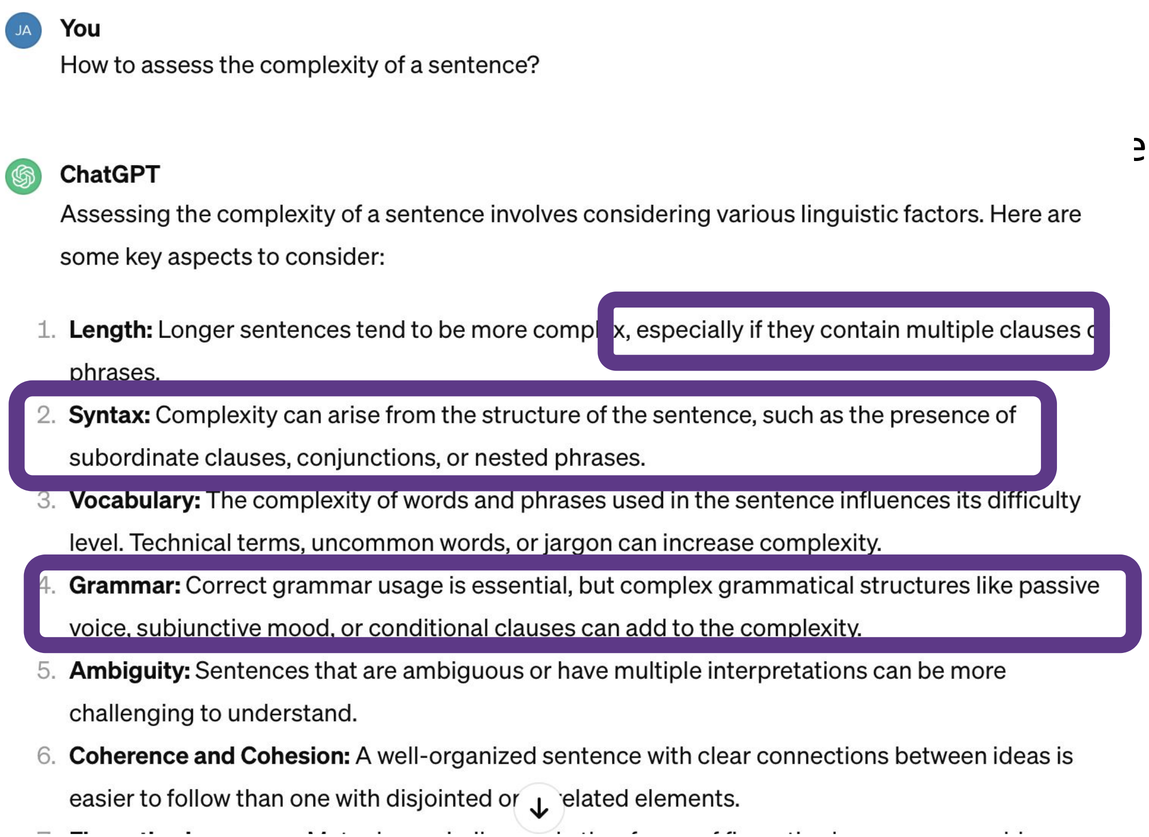
A task in a recent BA thesis: Use LLMs to translate standard text into simple language.

But what defines simple language?

One (recent) Example

A task in a recer … e language.

But what define



You
How to assess the complexity of a sentence?

ChatGPT
Assessing the complexity of a sentence involves considering various linguistic factors. Here are some key aspects to consider:

1. **Length:** Longer sentences tend to be more compl x, especially if they contain multiple clauses c phrases.
2. **Syntax:** Complexity can arise from the structure of the sentence, such as the presence of subordinate clauses, conjunctions, or nested phrases.
3. **Vocabulary:** The complexity of words and phrases used in the sentence influences its difficulty level. Technical terms, uncommon words, or jargon can increase complexity.
4. **Grammar:** Correct grammar usage is essential, but complex grammatical structures like passive voice, subjunctive mood, or conditional clauses can add to the complexity.
5. **Ambiguity:** Sentences that are ambiguous or have multiple interpretations can be more challenging to understand.
6. **Coherence and Cohesion:** A well-organized sentence with clear connections between ideas is easier to follow than one with disjointed or ↓ elated elements.

Buffalo

Buffalo	buffalo	Buffalo	buffalo	buffalo	buffalo	Buffalo	buffalo.
NNP	NN	NNP	NN	VB	VB	NNP	NN

buffalo, *noun*

A species of North American bison

Buffalo, *proper noun*

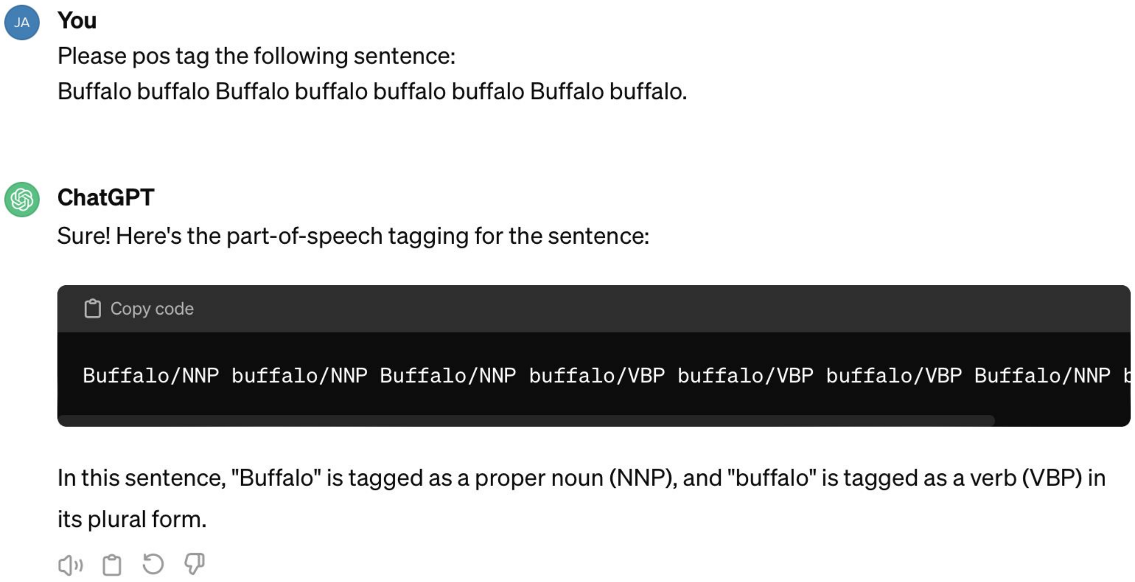
A city in the U.S. state of New York

buffalo, *verb*

To bully, to intimidate, to harass

Buffalo

Buffalo	buffalo	Buffalo	buffalo	buffalo	buffalo	Buffalo	buffalo.
NNP	NN	NNP	NN	VB	VB	NNP	NN



You

Please pos tag the following sentence:
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

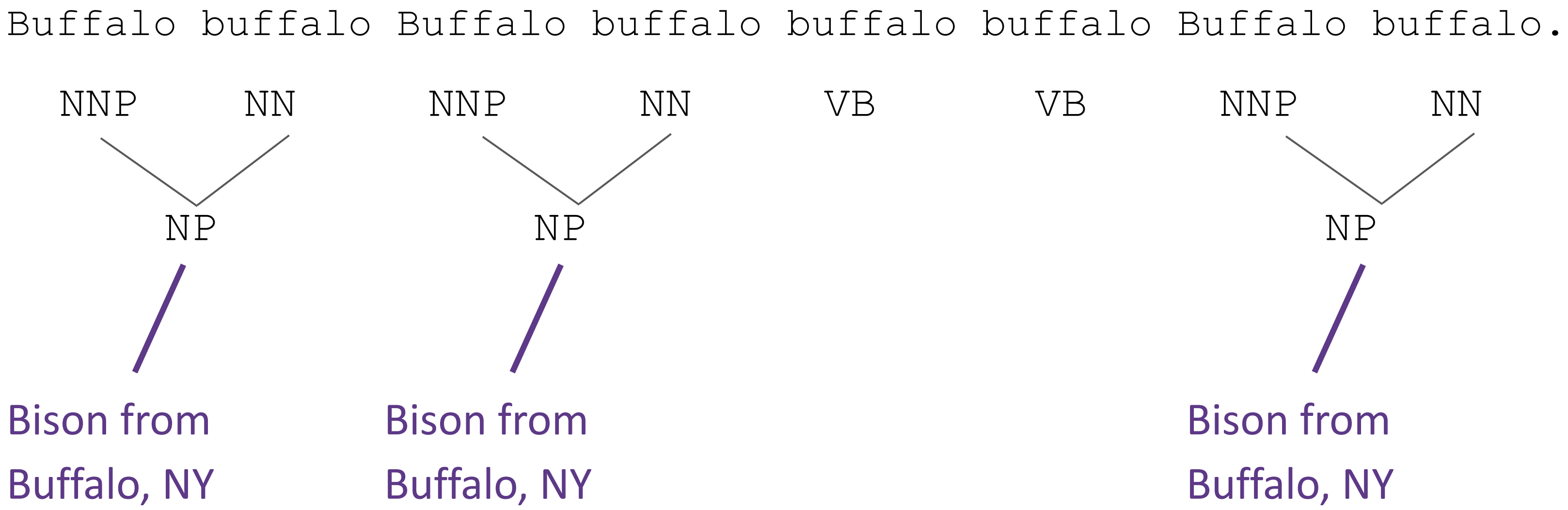
ChatGPT

Sure! Here's the part-of-speech tagging for the sentence:

```
Buffalo/NNP buffalo/NNP Buffalo/NNP buffalo/VBP buffalo/VBP buffalo/VBP Buffalo/NNP b
```

In this sentence, "Buffalo" is tagged as a proper noun (NNP), and "buffalo" is tagged as a verb (VBP) in its plural form.

Buffalo: Syntax Tree



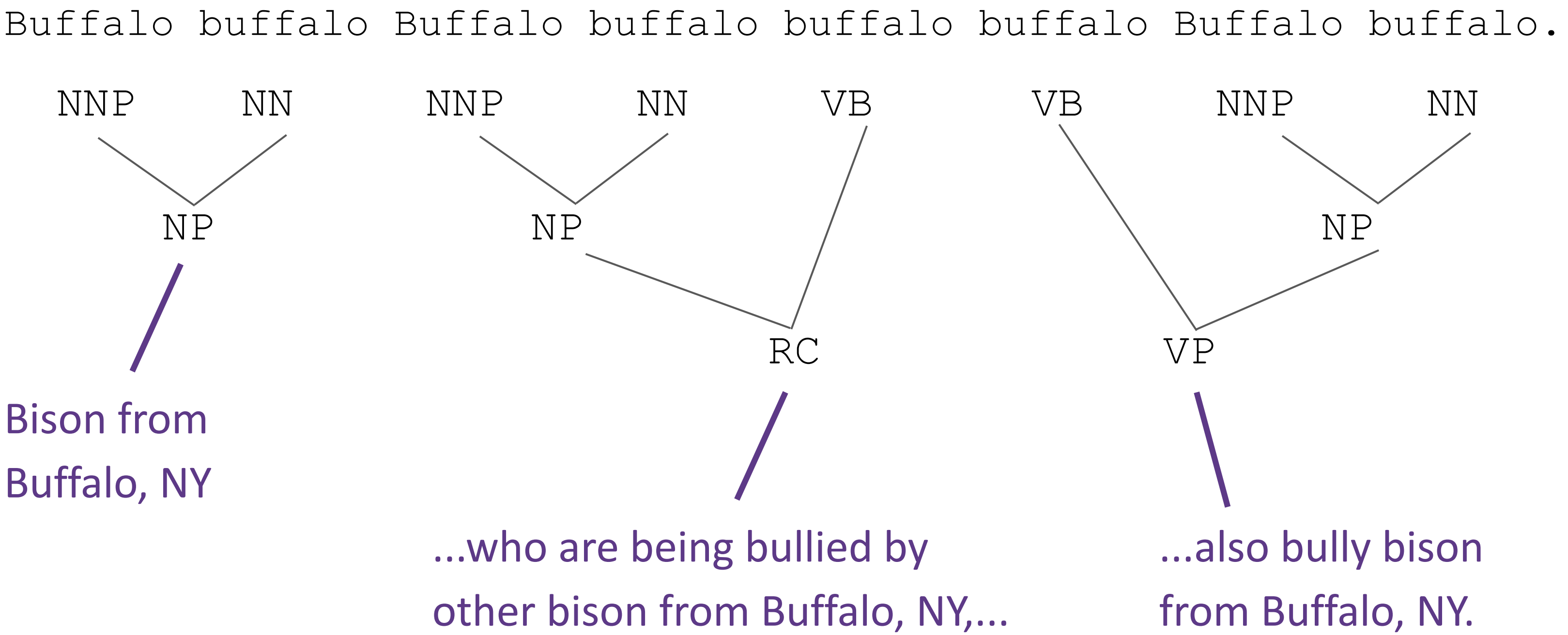
`Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.`

Bison from Buffalo, NY

Bison from Buffalo, NY

Bison from Buffalo, NY

Buffalo: Syntax Tree



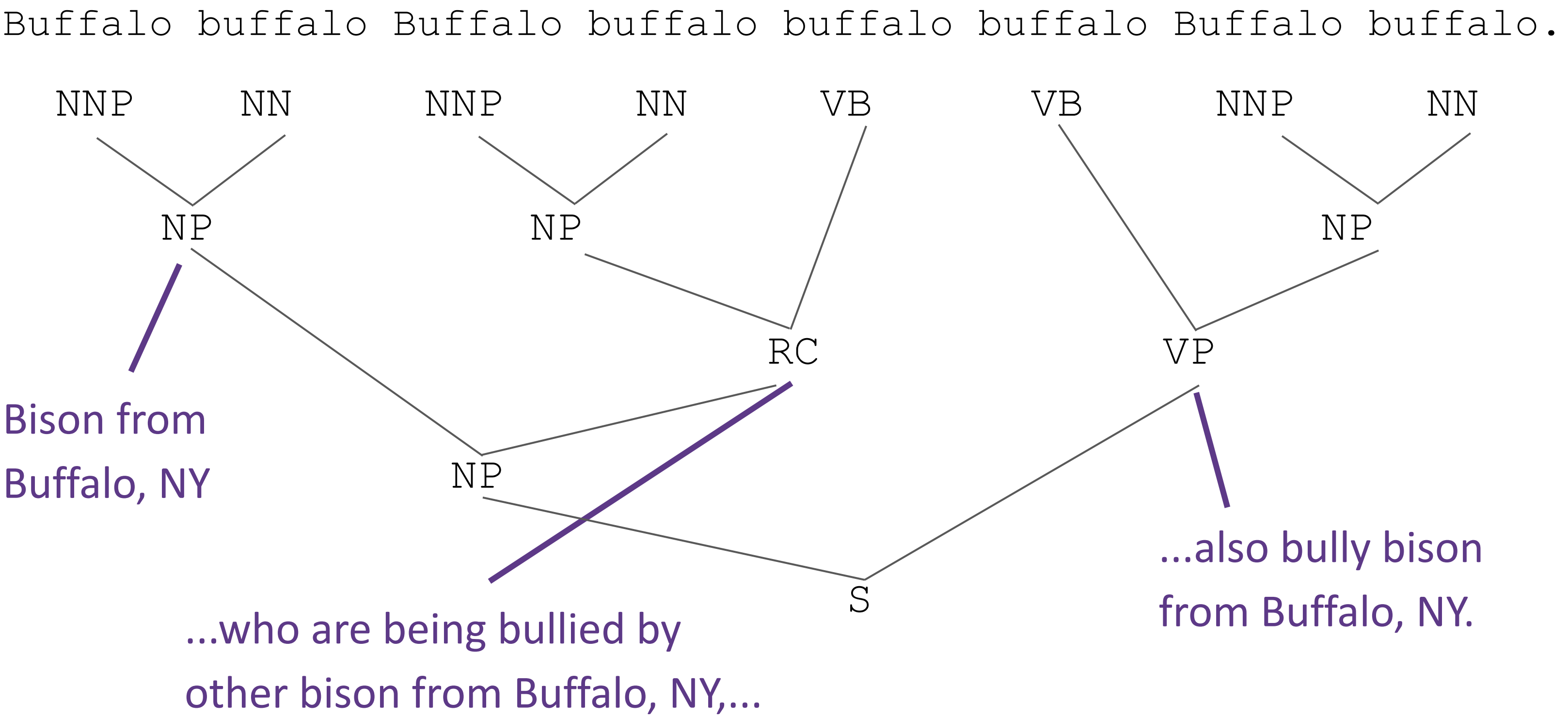
`Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.`

Bison from Buffalo, NY

...who are being bullied by other bison from Buffalo, NY,...

...also bully bison from Buffalo, NY.

Buffalo: Syntax Tree



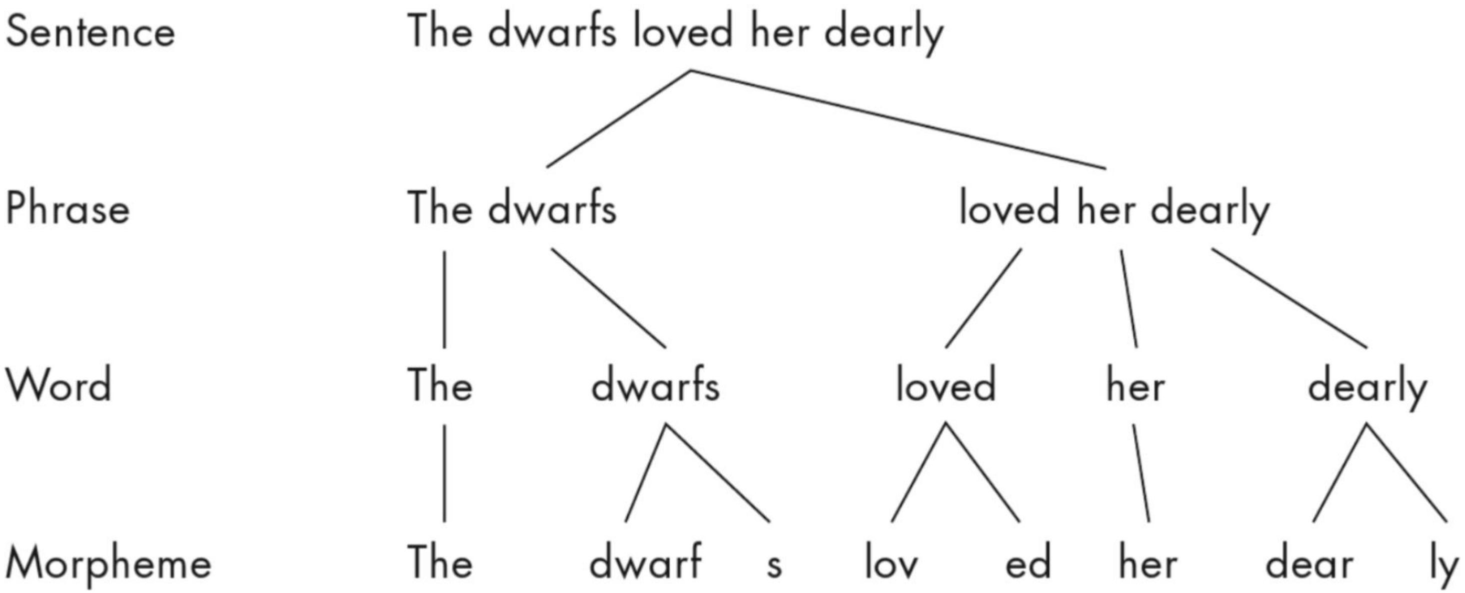
`Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.`

Bison from Buffalo, NY

...who are being bullied by other bison from Buffalo, NY,...

...also bully bison from Buffalo, NY.

What is Parsing?



Sentence The dwarfs loved her dearly

Phrase The dwarfs loved her dearly

Word The dwarfs loved her dearly

Morpheme The dwarf s lov ed her dear ly

Parsing is the process of determining the **syntactical structure** of a sentence with respect to a given **grammar** (from which the sentence was generated).

Syntax

Language is more than a "bag of words"!

- There are grammatical rules.
- Grammatical rules do not apply to individual words…
- ...but rather to categories of words.
- Words in these Categories are interchangeable.
- When we learn a new word, we learn its syntactical usage!

Example:

- A simple sentence includes just a **subject** and a **predicate**.
- The subject is a **noun phrase** and the predicate is a **verb phrase**:
  - Noun phrase: `The cat, Samantha, She`
  - Verb phrase: `arrived, went away, had dinner`

Syntactical Correctness ≠ Meaningful Semantics

`Colorless green ideas sleep furiously.`

vs.

`*Furiously sleep ideas green colorless.`

Syntactical Correctness ≠ Meaningful Semantics

`Colorless green ideas sleep furiously.`

vs.

`*Furiously sleep ideas green colorless.`

Composed by Noam Chomsky (1957) as an example of a sentence that is grammatically well-formed, but semantically nonsensical.

"It is fair to assume that neither sentence (1) nor (2) […] had ever previously occurred in an English discourse. Hence, in any statistical model that accounts for grammaticality, these sentences will be ruled out on identical grounds as equally "remote" from English. Yet (1), though nonsensical, is grammatical, while (2) is not grammatical."

https://upload.wikimedia.org/wikipedia/commons/c/cf/Noam_Chomsky_portrait_2017_retouched.png retouched by Wugapodes, CC BY-SA 4.0

But Semantics are Flexible

Colorless green ideas sleep furiously.

vs.

*Furiously sleep ideas green colorless.

A submission to a literary competition at Stanford Unive :
“It can only be the thought of verdure to come, which prompts us in the autumn to buy these dormant white lumps of vegetable matter covered by a brown papery skin, and lovingly to plant them and care for them. It is a marvel to me that under this cover they are labouring unseen at such a rate within to give us the sudden awesome beauty of spring flowering bulbs. While winter reigns the earth reposes but these ***colourless green ideas sleep furiously****.”* (C.M. Street, 1985)

Formal Grammars

Formal Grammars

A formal grammar `G = (N,T,P,S)` provides a schema for generating a formal language. It consists of:

- A finite set of **non-terminal symbols** `N` (for natural language: constituents)
- A finite set of **terminal symbols** `T` (for natural language: words)
- A **start symbol** `S` (for natural language: a sentence)
- A finite set of **production rules** `P` of the form: left hand side ⟶ right hand side

Formal Grammars

The grammar's language `L(G)` is the set of all sequences of terminal symbols that can be generated with the grammar. To generate a sentence from the language:

- Beginning with the start symbol, consecutively apply rules ("derivation")
- Applying a rule replaces all symbols on the left hand side with the right hand side.
- Once only terminal symbols remain, the derivation is complete

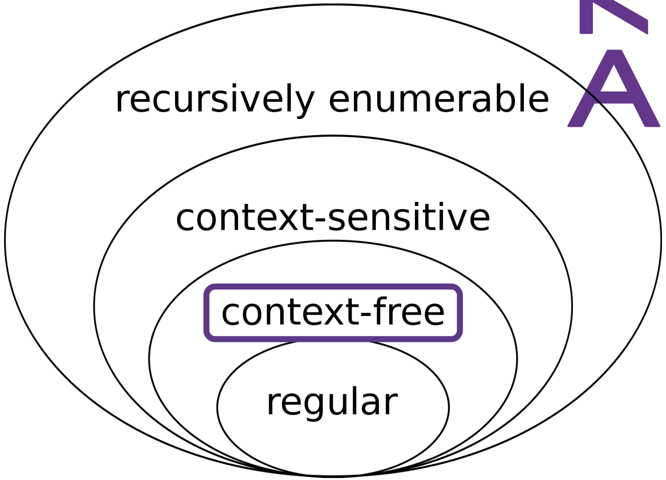
The Chomsky Hierarchy

Four types of languages:

- Type-0: Recursively enumerable languages
  Rules have the form: $\gamma \rightarrow \alpha$ (no constraints)
- Type-1: Context-sensitive languages
  Rules have the form: $\alpha A \beta \rightarrow \alpha \gamma \beta$
- Type-2: Context-free languages
  Rules have the form: $A \rightarrow \alpha$
- Type-3: Regular languages
  Rules have the form: $A \rightarrow a$ or $A \rightarrow aB$

Where:

a is a terminal symbol. A, B are non-terminal symbols. α, β, γ are arbitrary sequences of terminal / non-terminal symbols. α and β maybe empty. γ cannot be empty.



Natural languages can be derived from (mostly) context free grammars (CFGs).

Chomsky Normal Form

A context-free grammar (CFG), is said to be in Chomsky normal form if all of its production rules are in one of the following three forms:

- $A \rightarrow BC$
- $A \rightarrow a$
- $S \rightarrow \varepsilon$

where:

- A, B, C are nonterminal symbols
- a is a terminal symbol
- S is the start symbol
- ε denotes the empty string
- Neither B nor C can be S

Not all CFGs satisfy these requirements. However, for any CFG, there exists an equivalent grammar in Chomsky normal form. Thus, grammars can be normalized.

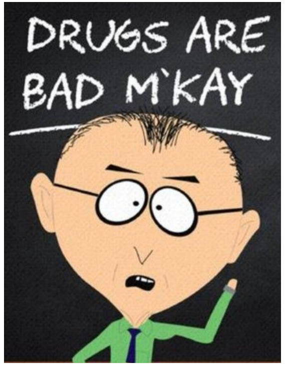
Further details: → Assignment

Phrase Structure

Constituency

Phrase structure organizes words into nested **constituents**. But what is a constituent?

- A word or a group of words that functions as a single unit within a hierarchical structure.
- A constituent behaves as a unit that can appear in different places.



Example:

- `Mr. Mackey talked [to the children] [about drugs].`
- `Mr. Mackey talked [about drugs] [to the children].`
- `*Mr. Mackey talked drugs to the children about.`

A * in examples typically denotes ungrammatical or unobserved sentences.

Constituents

Consider how we can generate sentences `(S)`:

- A simple grammar for a simple sentence: e.g., `“Birds fly”`
    - `S → NN VB`
    - `NN → birds | planes | superheroes`
    - `VB → fly | sing | fall`
- Other sample sentences: `“Birds sing”`, `“Superheroes fly”`, etc.

Limitation: In the above examples, verbs are **intransitive** (i.e., have no direct object)

- Example: `superheroes save [who | what]`
- We need **transitive verbs** and **direct objects**
- We need determiners, e.g., `“the mayor”, “the day”`

Noun Phrases (`NP`)

Consider an extended grammar:

  - `S → NP VP`
  - `NP → DT NN`
  - `VP → VB NP`
  - `DT → the | a`
  - `NN → child | firefighter | superhero`
  - `VB → likes | sees | saves`
- Example sentence: “`A firefighter saves the child`”
- **Limitation:** This grammar only allows noun phrases starting with a determiner. What about proper nouns like “`Gotham City`”?
- **Solution:** Add a rule to allow different expansions of NPs: `NP → NNP | DT NN`

Noun Phrases: Updated Grammar

Consider the updated grammar:

  - `S → NP VP`
  - `NP → DT NN`
  - **`NP → NNP`**
  - `VP → VB NP`
  - `DT → the | a`
  - `NN → child | firefighter | superhero`
  - **`NNP → Gotham City | New York`**
  - `VB → likes | sees | saves`
- Example sentence: “`The superhero saves Gotham City`”

Noun Phrases: Adjectives

What about adjectives?

- Adjectives modify nouns
- Wherever `NN` is allowed in a sentence, we could also use `JJ NN`, `DT JJ NN`, etc.
- This also works for proper nouns, e.g., “`beautiful New York`”

Introduce new rules:

- `NP → NN | DT NN | JJ NN | DT JJ NN | JJ NNP`
- Notation: Optional categories can be also marked using parentheses
    - `NP → (DT) (JJ) NN`

Verb Phrases (VP)

Structures of verb phrases:

- Verb phrases can be intransitive: `“Samantha ran”`
- Verb phrases can be intransitive with a **prepositional phrase**: `“Samantha ran` **`in Central Park`**`”`
- Verb phrases can contain a **particle**: `“Samantha ran` **`away`**`”`
- Verb phrases can be transitive: **`“Samantha bought a cookie”`**
- Verb phrases can be transitive with a **prepositional phrase**: `“Samantha bought a cookie` **`for John`**`”`

We can combine the above options in an overall structure:

- `VP -> VB (NP)` $\boxed{\texttt{(IN (NP))}}$ **prepositional phrase (PP)**

Verb Phrases: Updated Grammar

Consider the updated grammar:

- `S → NP VP`
- **`NP → (DT) (JJ) NN`**
- `NP →` **`(DT) (JJ)`** `NNP`
- `VP → VB` **`(NP) (IN (NP))`**
- `DT → the | a`
- `NN → child | firefighter | superhero | villain`
- `NNP → Gotham City | New York`
- `VB → likes | sees | saves`
- **`JJ → scary | fast`**
- **`IN → from | away`**

- Example sentence:

  `“The superhero saves the child from the scary villain”`

Prepositional Phrases `(PP)`

Structures of prepositional phrases:

- `Mary bought a book` **`for John`** **`in a bookstore`**
- `The bookstore` **`on Main Street`** `sells magazines`
- `Mary ran` **`away`**
- `Mary ran` **`down the hill`**

Changes are needed to both `NP` and `VP` in our grammar to accommodate `PP`:

- Wherever a preposition is allowed, it can be followed by a noun phrase
- `Run up` vs. `run up the street`
- `NP` can contain any number of `PP`

Prepositional Phrases: Updated Grammar

We need to change our grammar accordingly:

- S → NP VP
- NP → (DT) (JJ) NN **(PP)**
- NP → (DT) (JJ) NNP
- VP → VB (NP) **(PP)**
- **PP → IN (NP)**
- DT → the | a
- NN → child | firefighter | superhero | villain
- NNP → Gotham City | New York | U.S. | Earth
- VB → likes | sees | saves
- JJ → scary | fast
- IN → from | away | in

Due to the recursion NP → PP → NP, we can produce arbitrarily long sentences

- “The superhero [from New York] [in the U.S.] [on Earth]...”

Further Natural Language Grammar Concepts

Repetition of **adjectives** `(JJ*)`

- The **Little Red Riding** Hood

Nested sentences with **subordinating conjunction**

- VP → VB (NP) (**C S**) (PP)
- I forgot **whether I took the dog out**

Combination of sentences with **coordinating conjunctions**:

- NP → NP **CC** NP; **e.g.,** apples **and** oranges
- PP → PP **CC** PP; **e.g.,** smart **yet** naive
- VP → VP **CC** VP; **e.g.,** I like walking **or** running

Constituency Parsing: Task and Challenges

Parsing: Task Definition

Given a grammar (CFG) and an input sentence, constituency parsing associates a tree structure with the phrase structure of the sentence that explains the derivation rules.

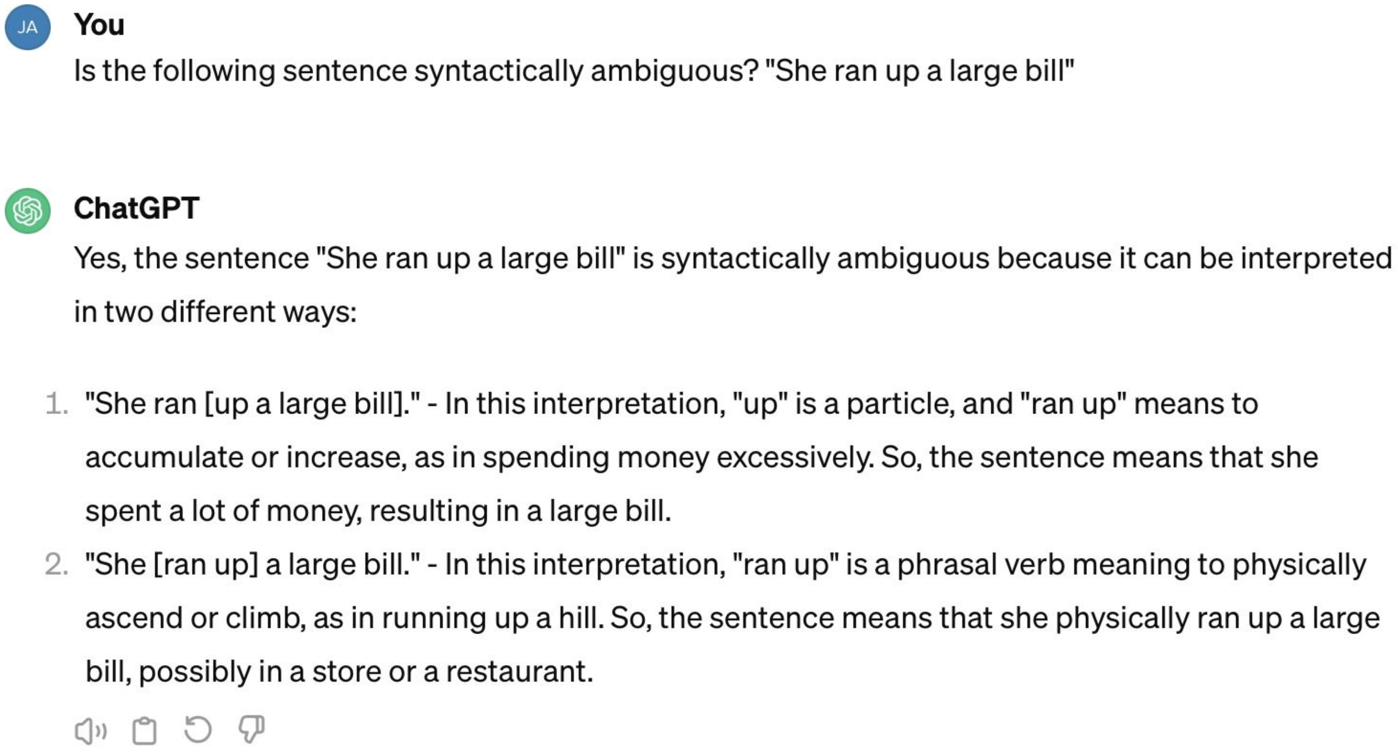
- There may be exactly one matching tree structure
- There may be many matching tree structures
- There may be none

Parsing Challenges: Syntactic Ambiguities

Syntactic ambiguities can results in semantically different but syntactically correct tree structures:

- Prepositional phrase attachment
  `The boy saw the woman with the telescope.`
- Coordination scope
  `Small boys and girls are playing.`
- Particles vs. prepositions
  `She ran up a large bill.`
- Gerund vs. adjectives
  `Frightening kids can cause trouble.`

My English is not that good, where is the ambiguity?



You

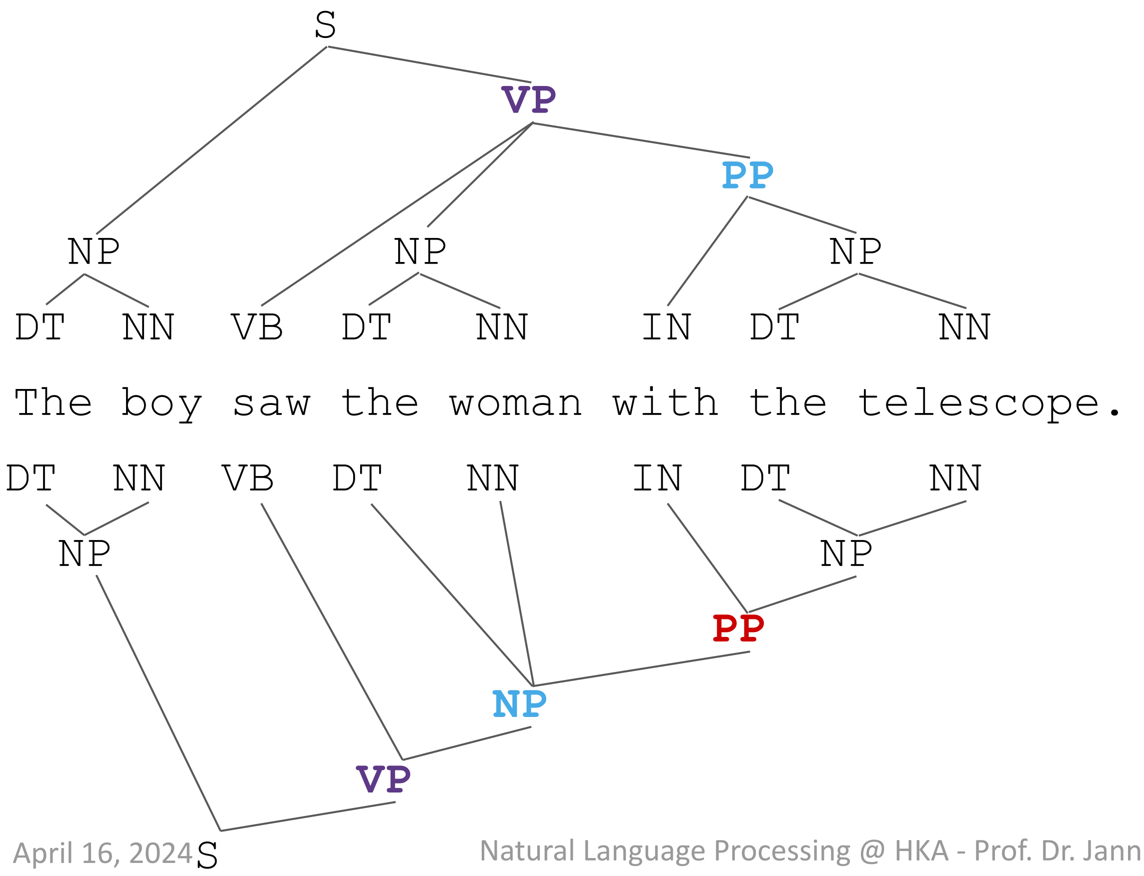
Is the following sentence syntactically ambiguous? "She ran up a large bill"

ChatGPT

Yes, the sentence "She ran up a large bill" is syntactically ambiguous because it can be interpreted in two different ways:

1. "She ran [up a large bill]." - In this interpretation, "up" is a particle, and "ran up" means to accumulate or increase, as in spending money excessively. So, the sentence means that she spent a lot of money, resulting in a large bill.
2. "She [ran up] a large bill." - In this interpretation, "ran up" is a phrasal verb meaning to physically ascend or climb, as in running up a hill. So, the sentence means that she physically ran up a large bill, possibly in a store or a restaurant.

Syntactic Ambiguity Parsing Example



Derivation:

- S → NP **VP**
- **VP** → VB NP PP

Derivation:

- S → NP **VP**
- **VP** → VB **NP**
- **NP** → DT NN **PP**

Parsing Challenges: Agreement

Our “simple” grammar might already feel complex, but it is still very simplistic.
In reality, we need rules to adjust for agreement:

- Number (plural, singular)
  `Samantha` **`is`** `/ people` **`are`**
- Person (first, second, third,...)
  `I` **`am`** `/ you` **`are`** `/ Samantha` **`is`**
- Tense (present, past,...)
  `Samantha` **`was`** `reading / Samantha` **`is`** `reading`

Parsing Challenges: Agreement

And English is quite simple. Other languages are more complex and include, e.g.:

- Case (nominative, dative, genitive...)
  `Das` `Brot / ` **`dem`** `Brot / ` **`des`** `Brots`
- Gender (female, male, neutral,...)
  `Le` `monde / ` **`la`** `lune`

Constituency Parsing Algorithms

Parsing Algorithms: Creating Parse Trees

A **parse tree** (or derivation tree) is an ordered rooted tree that graphically represents the syntactic information of a string derived from a (context-free) grammar. There are two different approaches to generate a parse tree:

Top-down approach

- Starts with the starting symbol `S`
- Iteratively moves down towards tree leaves using productions
- Problem: Explores options that won‘t lead to a full parse

Bottom-up approach

- Starts from tree leaves (= tokens)
- Proceeds upward to the root (= the starting symbol `S`)
- Problem: Explores options that don‘t match the full sentence (→ backtracking)

Algorithm: Leftmost Derivation

Leftmost derivation is not a specific algorithm, but rather a class of algorithms. It is designed for restricted context free grammars and does not work perfectly on natural language.

Leftmost derivation is an example of **top-down** parsing.

Algorithm outline:

- Start with `S`
- In each step, apply a production rule to the leftmost non-terminal symbol
- Produce the observed terminal symbols (= tokens)
- Terminates once all observed words have been created.

Example:

- `S`
- **`[S → NP VP]`**
- **`[NP → DT NN]`** `VP`
- **`[DT → The]`** `NN VP`
- `The` **`[NN → apple]`** `VP`
- `The apple` **`[VP → ...]`**

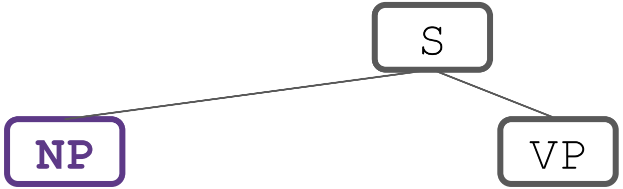
Leftmost Derivation: Example

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

`The child ate the cake with the fork.`

Leftmost Derivation: Example

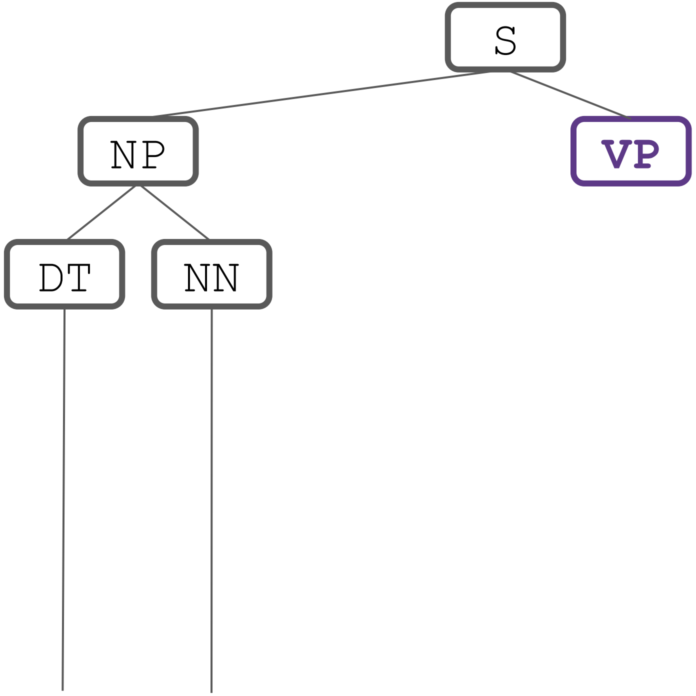


Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

The child ate the cake with the fork.

Leftmost Derivation: Example

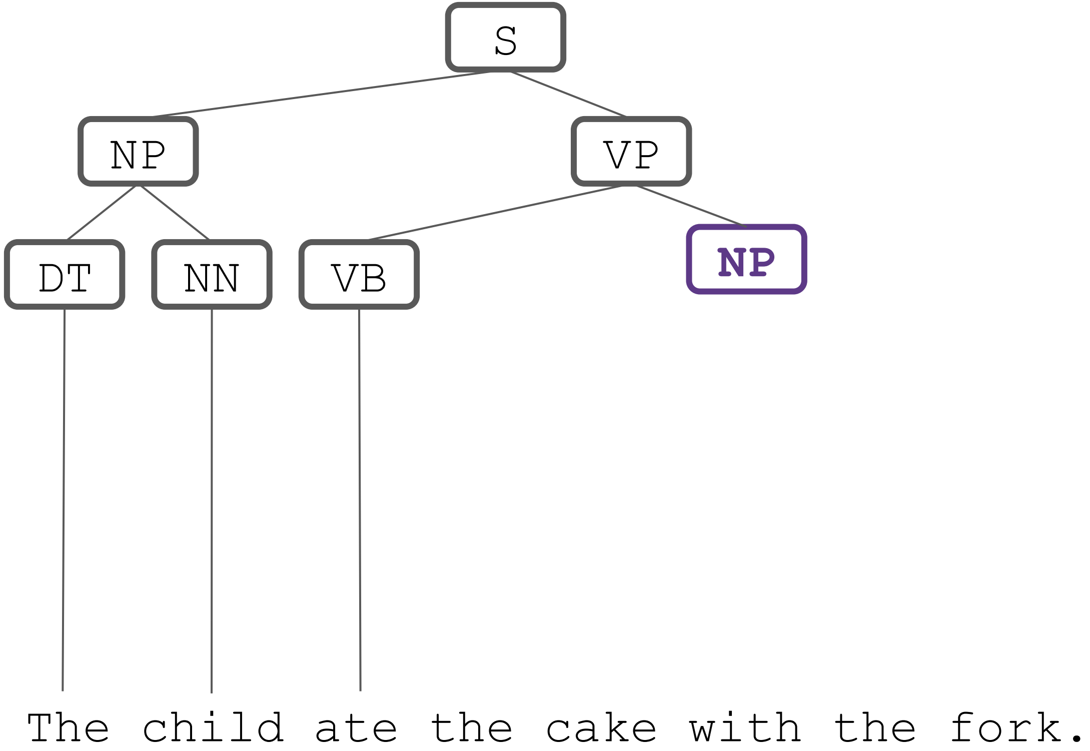


The child ate the cake with the fork.

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

Leftmost Derivation: Example



Example grammar:

S → NP VP

NP → DT NN | **NP PP**

PP → IN NP

VP → VB NP | VP PP

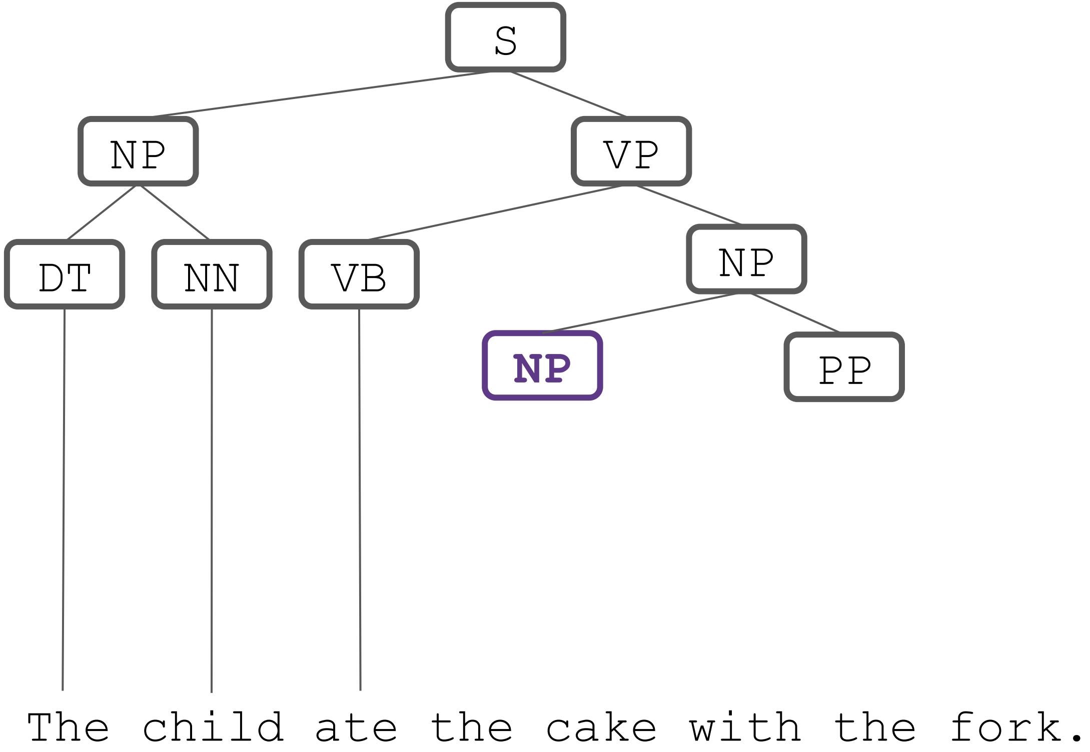
DT → the

NN → child | cake | fork

VB → ate

IN → with

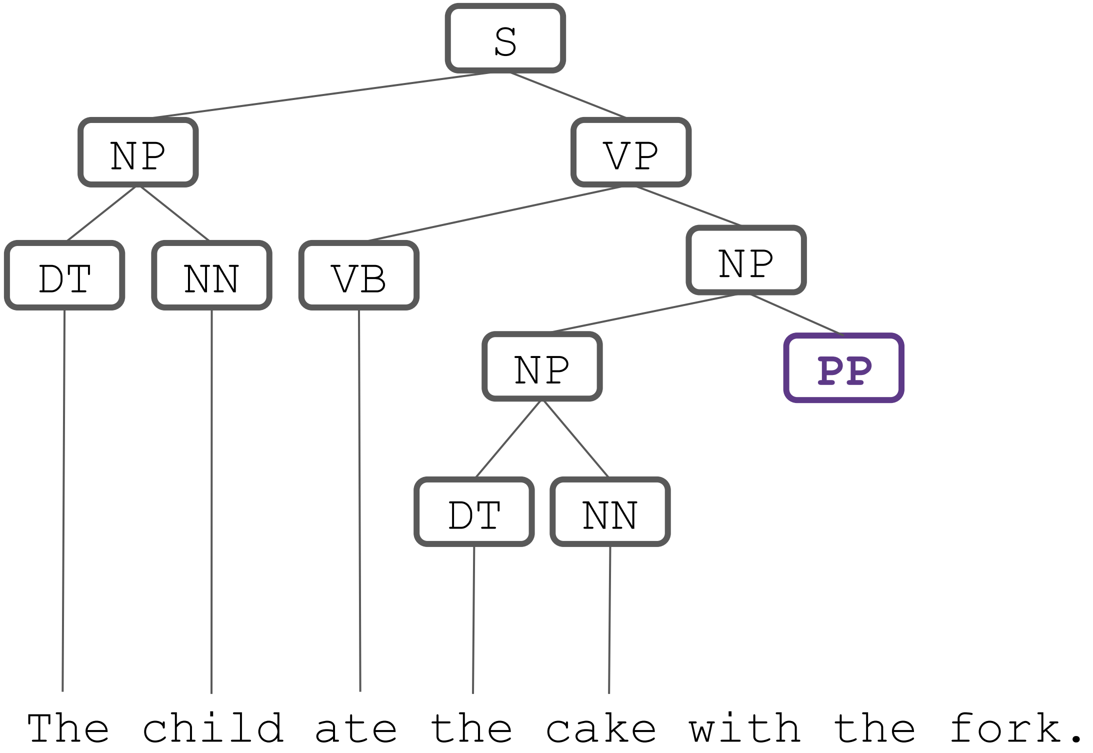
Leftmost Derivation: Example



Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

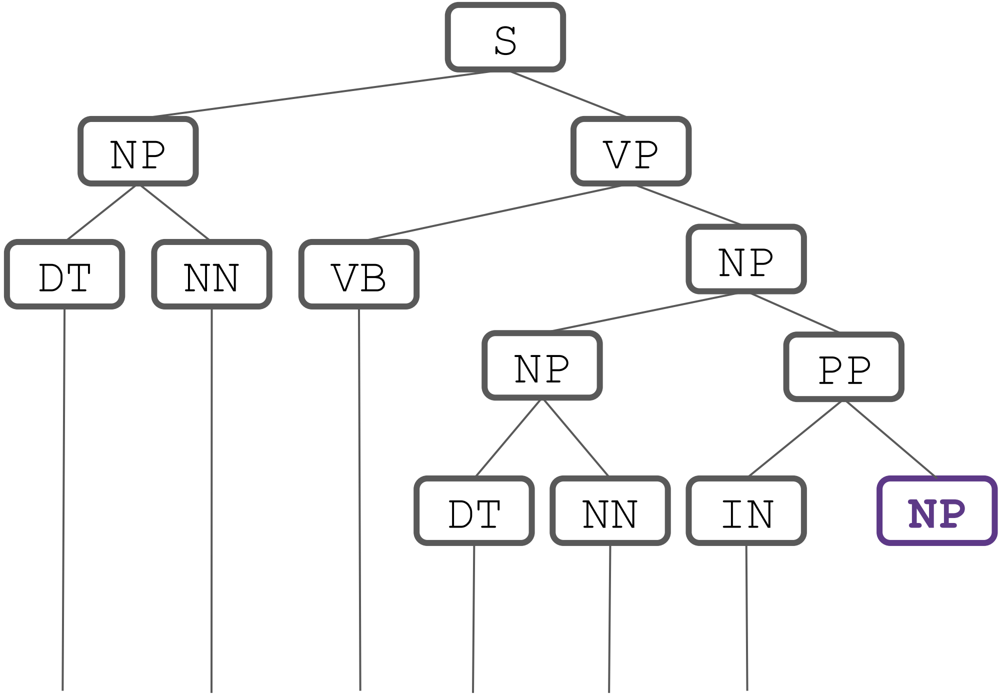
Leftmost Derivation: Example



Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

Leftmost Derivation: Example

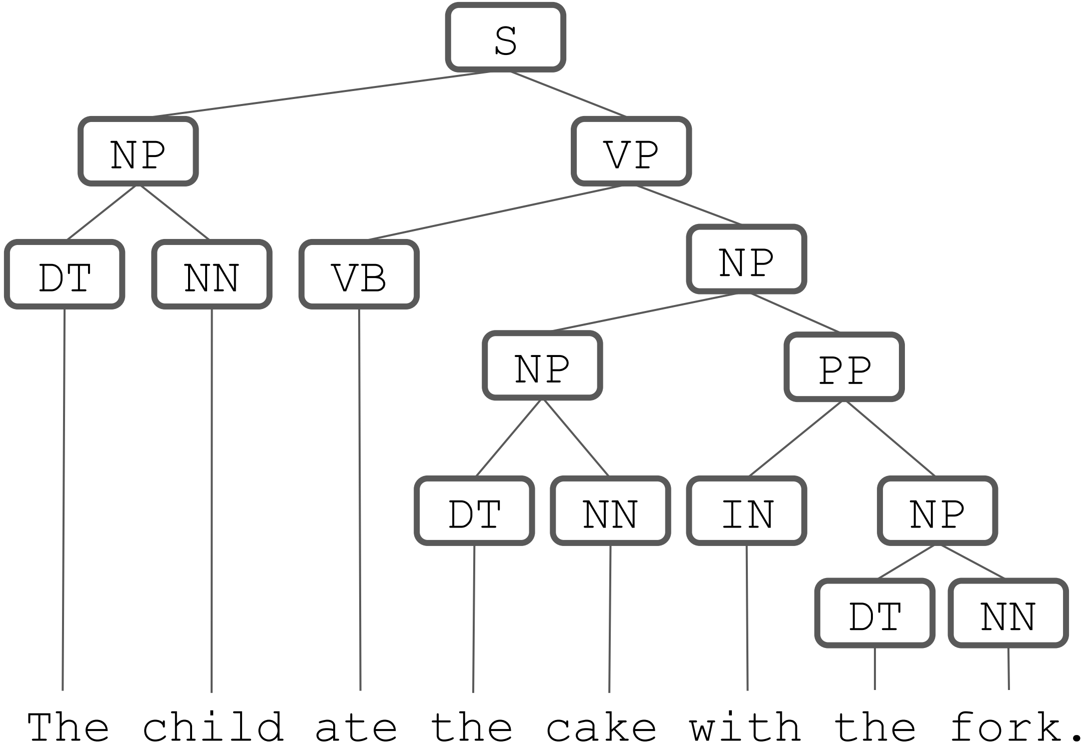


The child ate the cake with the fork.

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake | fork
VB → ate
IN → with

Leftmost Derivation: Example



Example grammar:

`S → NP VP`

`NP → DT NN | NP PP`

`PP → IN NP`

`VP → VB NP | VP PP`

`DT → the`

`NN → child | cake | fork`

`VB → ate`

`IN → with`

Algorithm: Shift Reduce Parser

A shift reduce parser is effectively a **pushdown automaton** that uses a stack for recognizing a context free language. It is an example of **bottom-up** parsing.

Algorithm outline:

- SHIFT operation:
  Push a word from the input sentence onto the stack.
- REDUCE operation:
  If the top `n` words on the top of the stack match the right hand side of a production rule, then they are popped and replaced by the left hand side of the production.
- STOPPING condition:
  The process stops when the input sentence has been processed and `S` has been popped from the stack.

Shift Reduce Parser: Example

Input and Stack: Operation

`[] ↓ the child ate the cake`

Example grammar:

```
S  → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake
VB → ate
```

↓: current position

[ ]: Stack content

Shift Reduce Parser: Example

Input and Stack:	Operation
[] ↓ the child ate the cake	SHIFT
[**the**] ↓ child ate the cake	**REDUCE**
[DT] ↓ child ate the cake	

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake
VB → ate

↓: current position
[ ]: Stack content

Shift Reduce Parser: Example

Input and Stack:	Operation
`[] ↓ the child ate the cake`	SHIFT
`[the] ↓ child ate the cake`	REDUCE
`[DT] ↓ child ate the cake`	SHIFT
`[DT` **`child`**`] ↓ ate the cake`	**REDUCE**
`[`**`DT NN`**`] ↓ ate the cake`	**REDUCE**
`[NP] ↓ ate the cake`	

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake
VB → ate

↓: current position
[ ]: Stack content

Shift Reduce Parser: Example

Input and Stack:	Operation
[] ↓ the child ate the cake	SHIFT
[the] ↓ child ate the cake	REDUCE
[DT] ↓ child ate the cake	SHIFT
[DT child] ↓ ate the cake	REDUCE
[DT NN] ↓ ate the cake	REDUCE
[NP] ↓ ate the cake	SHIFT
[NP **ate**] ↓ the cake	**REDUCE**
[NP VB] ↓ the cake	SHIFT
[NP VB **the**] ↓ cake	**REDUCE**
[NP VB DT] ↓ cake	SHIFT
[NP VB DT **cake**] ↓	**REDUCE**
[NP VB DT NN] ↓	

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → **the**
NN → child | **cake**
VB → **ate**

↓: current position
[ ]: Stack content

Shift Reduce Parser: Example

Input and Stack:	Operation
[] ↓ the child ate the cake	SHIFT
[the] ↓ child ate the cake	REDUCE
[DT] ↓ child ate the cake	SHIFT
[DT child] ↓ ate the cake	REDUCE
[DT NN] ↓ ate the cake	REDUCE
[NP] ↓ ate the cake	SHIFT
[NP ate] ↓ the cake	REDUCE
[NP VB] ↓ the cake	SHIFT
[NP VB the] ↓ cake	REDUCE
[NP VB DT] ↓ cake	SHIFT
[NP VB DT cake] ↓	REDUCE
[NP VB **DT** **NN**] ↓	**REDUCE**
[NP **VB** **NP**] ↓	REDUCE
[**NP** **VP**] ↓	**REDUCE**
[S] ↓	STOP

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → the
NN → child | cake
VB → ate

↓: current position
[ ]: Stack content

Algorithm: CKY Parser

The Cocke-Kasami-Younger parser (CKY parser, sometimes CYK parser) uses dynamic programming. It is an example of a **bottom-up** parser.

Core idea:

- Cache intermediate results (dynamic programming)
- For building a parse for a sequence `[i,j]`, keep track of all parses of subsequences `[i,k]` and `[k,j]` that are included in it ($i \leq k \leq j$).

Advantage:

- Backtracking: if the parser gets stuck, we can easily go back to a previous alternative

Downside:

- Requires a binarized grammar
- The grammar must be in **Chomsky Normal Form**

CKY Parser: Example

the	**DT**							
	child							
		ate						
			the					
				cake				
					with			
						the		
							fork	

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → a | the
NN → child | cake
NN → fork
VB → ate
IN → with

CKY Parser: Example

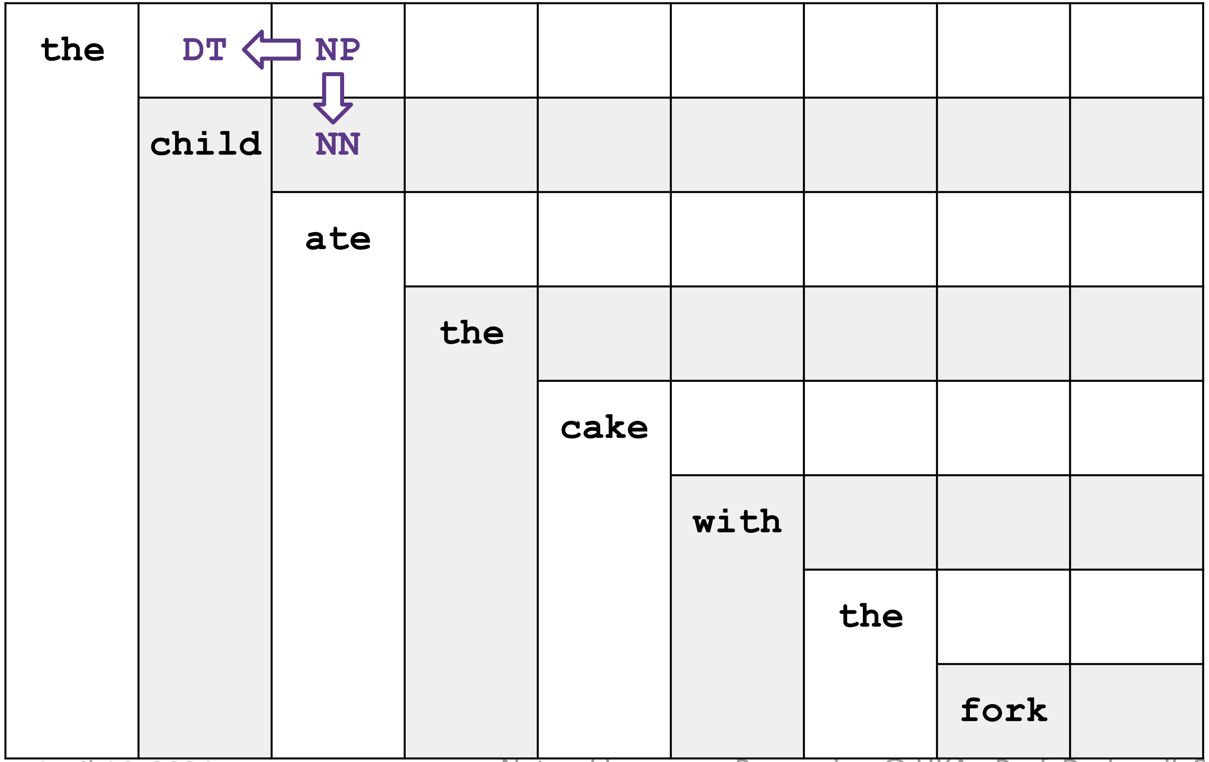
the	**DT**							
	child	**NN**						
		ate						
			the					
				cake				
					with			
						the		
							fork	

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP → VB NP | VP PP`
`DT → a | the`
`NN → child` `| cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Store backpointers: `NP` was created from `DT` and `NN`



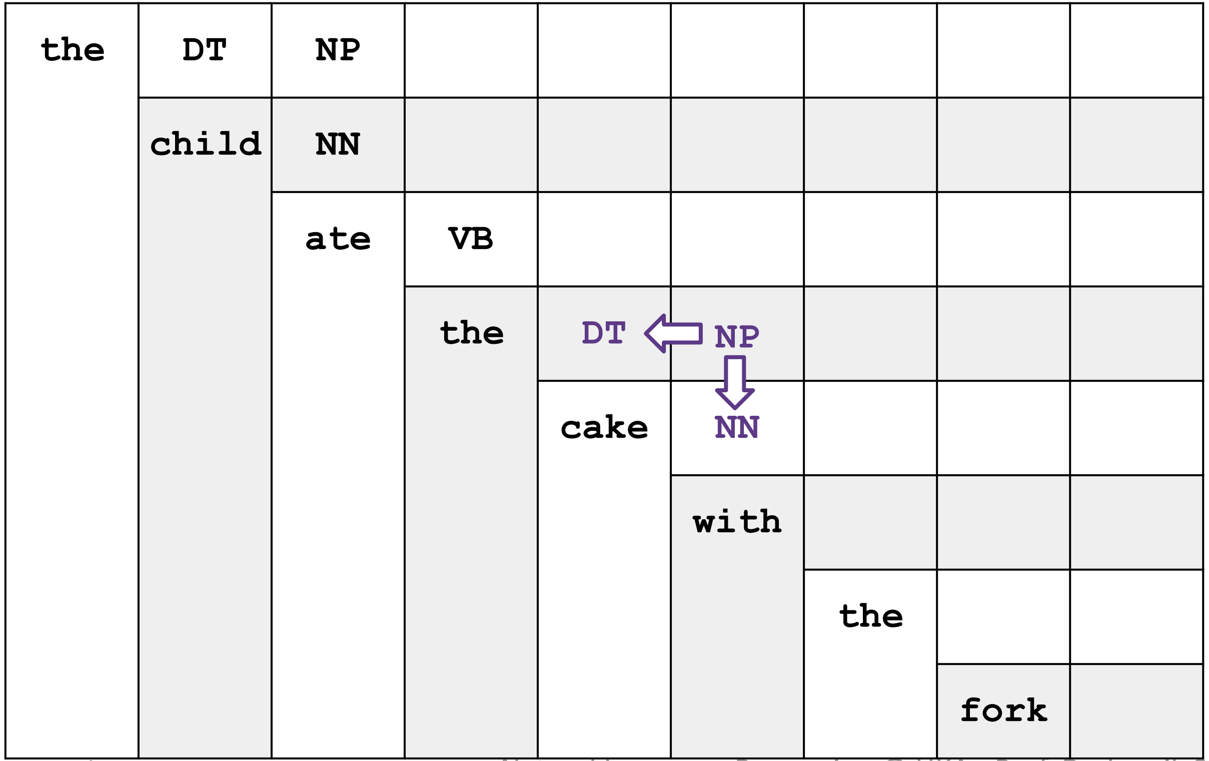
the	DT	NP						
	child	NN						
		ate						
			the					
				cake				
					with			
						the		
							fork	

Example grammar:

S → NP VP
NP → DT NN | NP PP
PP → IN NP
VP → VB NP | VP PP
DT → a | the
NN → child | cake
NN → fork
VB → ate
IN → with

CKY Parser: Example

Store backpointers: `NP` was created from `DT` and `NN`



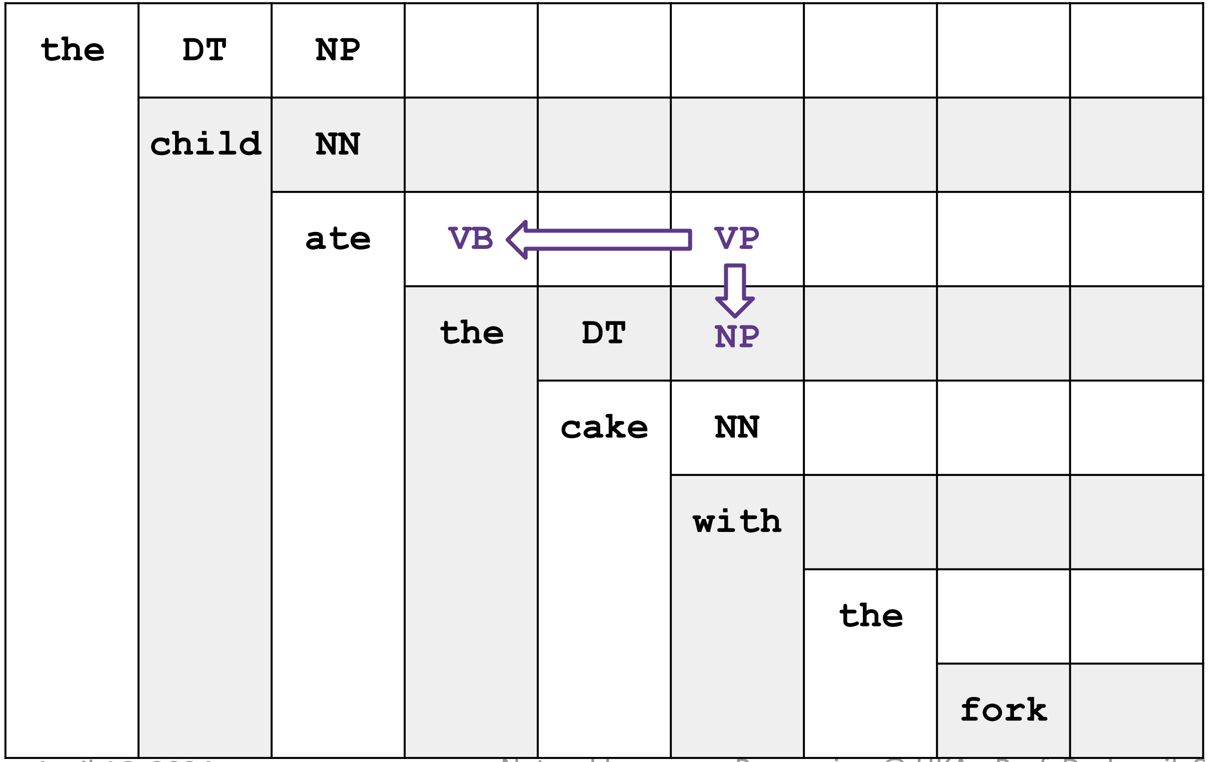
the	**DT**	**NP**						
	child	**NN**						
		ate	**VB**					
			the	**DT**	**NP**			
				cake	**NN**			
					with			
						the		
							fork	

Example grammar:

`S → NP VP`
`NP → DT NN` `| NP PP`
`PP → IN NP`
`VP → VB NP | VP PP`
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Store backpointers: `VP` was created from `VB` and `NP`



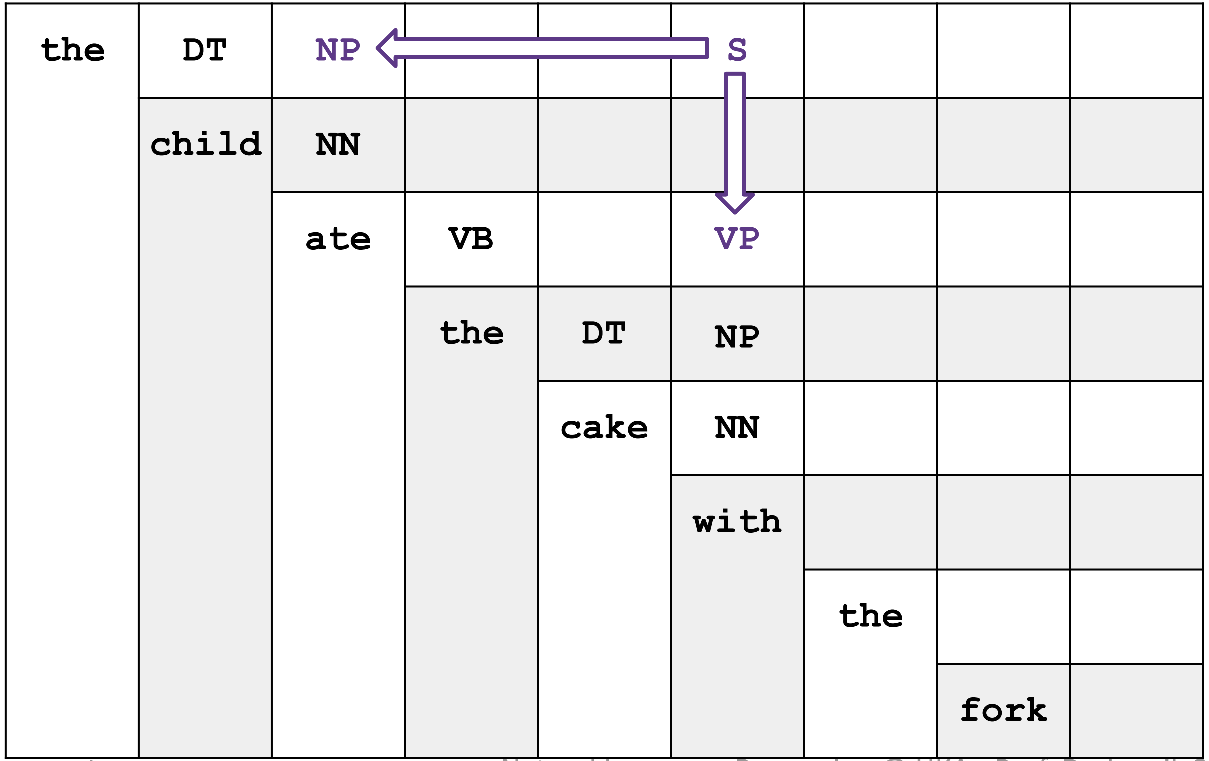
the	**DT**	**NP**						
	child	**NN**						
		ate	**VB**		**VP**			
			the	**DT**	**NP**			
				cake	**NN**			
					with			
						the		
							fork	

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP → VB NP` `| VP PP`
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Store backpointers: `S` was created from `NP` and `VP`



the	**DT**	**NP**			**S**			
	child	**NN**						
		ate	**VB**		**VP**			
			the	**DT**	**NP**			
				cake	**NN**			
					with			
						the		
							fork	

Example grammar:

S → NP VP

NP → DT NN | NP PP

PP → IN NP

VP → VB NP | VP PP

DT → a | the

NN → child | cake

NN → fork

VB → ate

IN → with

CKY Parser: Example

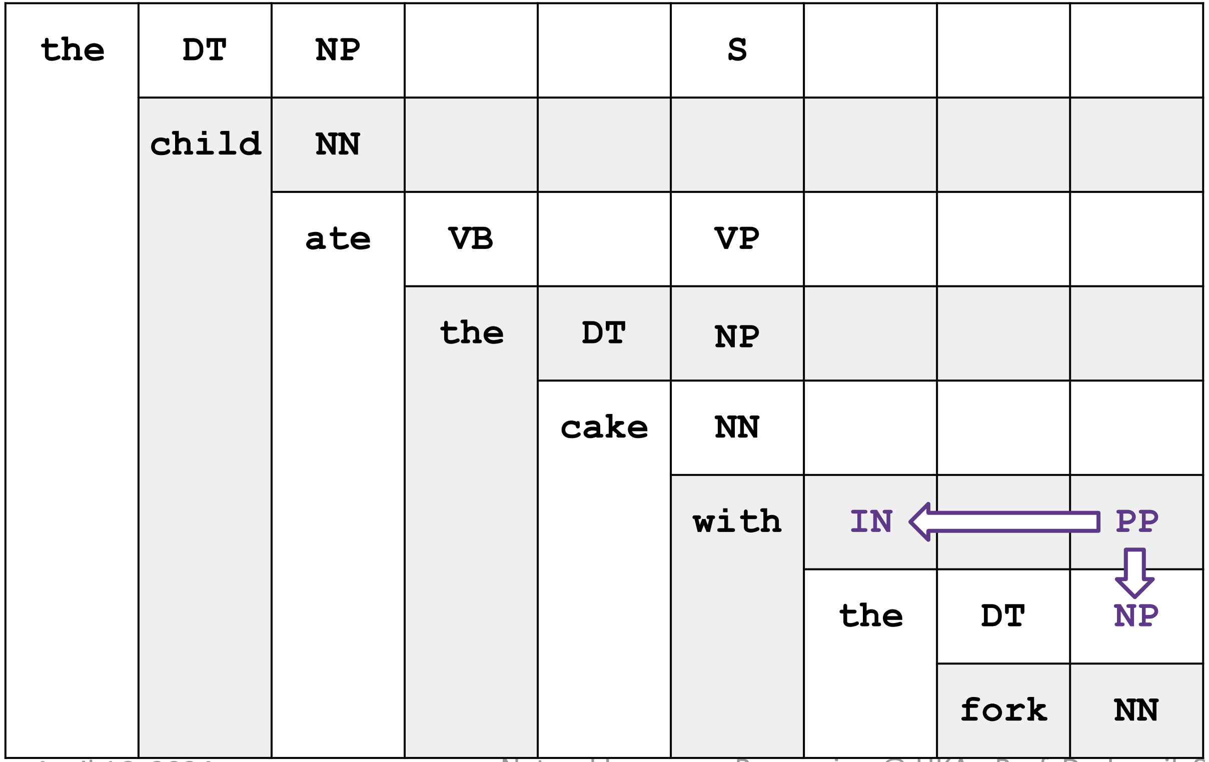
the	**DT**	**NP**			**S**			
	child	**NN**						
		ate	**VB**		**VP**			
			the	**DT**	**NP**			
				cake	**NN**			
					with	**IN**		
						the		
							fork	

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP → VB NP | VP PP`
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Store backpointers: `PP` was created from `IN` and `NP`



the	**DT**	**NP**			**S**			
	child	**NN**						
		ate	**VB**		**VP**			
			the	**DT**	**NP**			
				cake	**NN**			
					with	**IN**		**PP**
						the	**DT**	**NP**
							fork	**NN**

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP → VB NP | VP PP`
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Two alternative parses: **VP → VB NP** or **VP → VP PP**

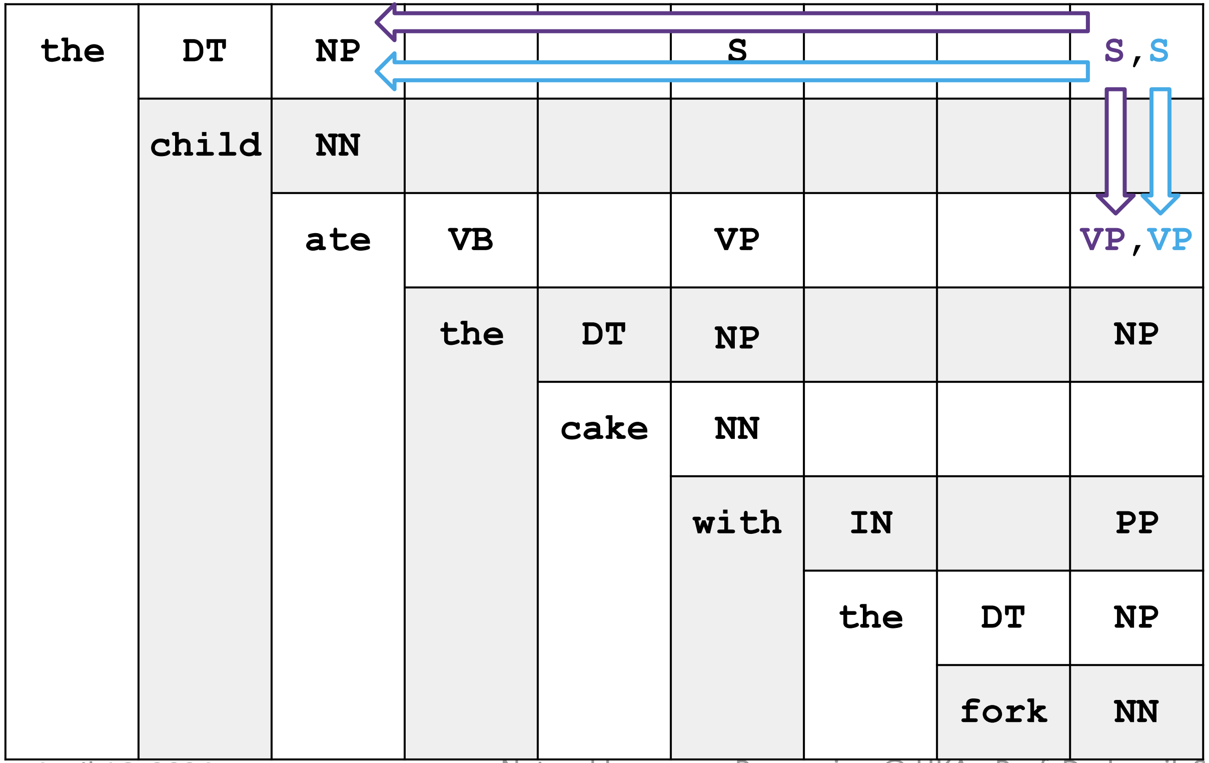
the	**DT**	**NP**			**S**			
	child	**NN**						
		ate	**VB**		**VP**			**VP,VP**
			the	**DT**	**NP**			**NP**
				cake	**NN**			
					with	**IN**		**PP**
						the	**DT**	**NP**
							fork	**NN**

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP →` **`VB NP`** `|` **`VP PP`**
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Example

Two alternative parses, based on the two alternative `VP`

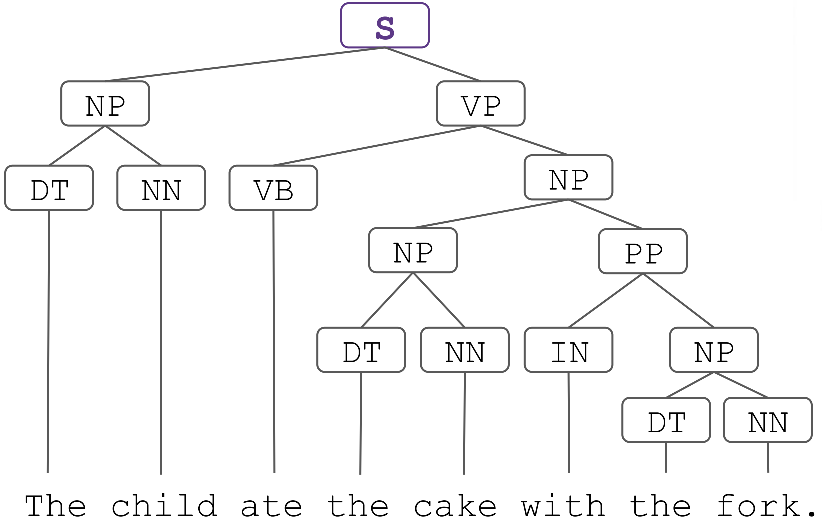


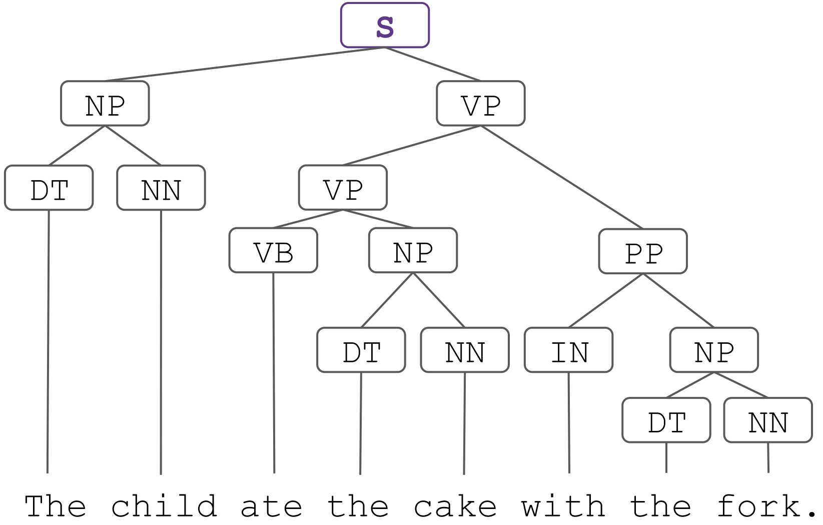
the	**DT**	**NP**			**S**			**S,S**
	child	**NN**						
		ate	**VB**		**VP**			**VP,VP**
			the	**DT**	**NP**			**NP**
				cake	**NN**			
					with	**IN**		**PP**
						the	**DT**	**NP**
							fork	**NN**

Example grammar:

`S → NP VP`
`NP → DT NN | NP PP`
`PP → IN NP`
`VP → VB NP | VP PP`
`DT → a | the`
`NN → child | cake`
`NN → fork`
`VB → ate`
`IN → with`

CKY Parser: Output





Statistical Parsing

How can we resolve ambiguous parse trees?

- Many parse trees are possible
- Some are more likely than others (very few people eat forks...)
- Let’s use corpus statistics to determine probable parse trees!

Here we go again...

Core idea:

- Use a grammar that attaches a probability to a production rule: **probabilistic context free grammar** (PCFG)
- All probabilities `p` of identical left-hand side symbols in rules must sum up to 1
- The probability of a parse tree is the product of all used production rules
- Probabilities can be learned from a labeled training corpus (treebank):

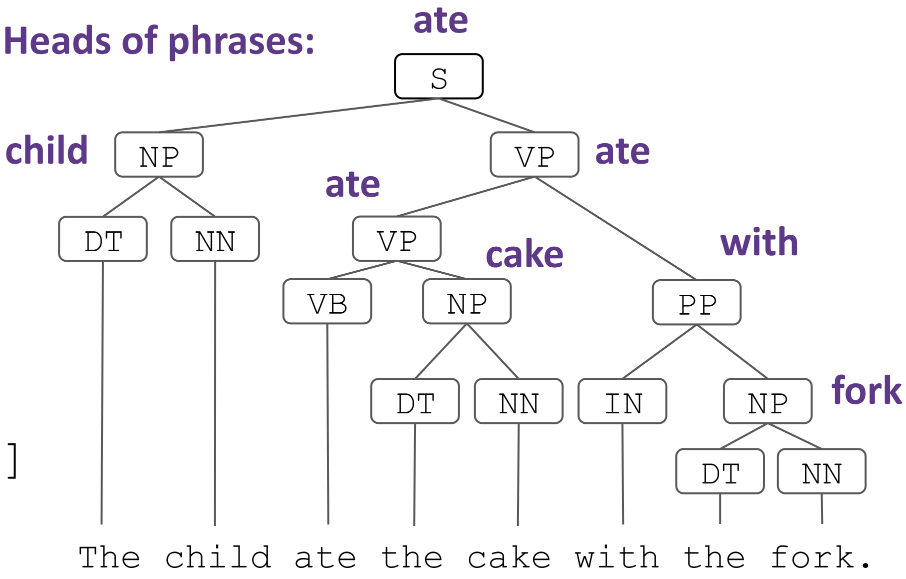
`p(S → NP VP) = count(S → NP VP) / count(S)`

Lexicalized Parsing

With corpus statistics, we can go further and construct a parser that takes lexical relations into account to compute probabilities.

Core idea:

- We do not just check the probability of rules, e.g. `VP → VP PP`
- Instead, we also check the probabilities of terminals, e.g., `VP[ate] → VP[ate] PP[with]`
- We can better model whether a specific `PP` belongs to a `VP` or `NP`



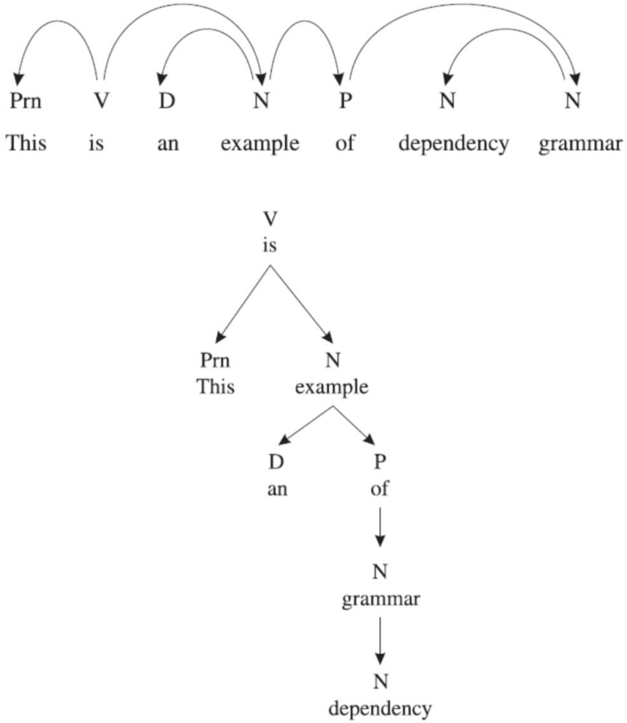
Dependency Parsing

(A Very Brief Glance)

Dependency Grammar

Dependency grammar assumes that syntactic structure consists only of dependencies between tokens.

- **Dependencies** are (labeled) asymmetric binary relations between two tokens.
- Dependencies typically form a tree with a single head (the root).



Dependency Parsing vs. Constituency Parsing

Dependency parsing:

- Models natural language via a dependency grammar
- Creates linguistic structure from the dependencies of tokens
- **Useful when we are interested in relations between words**

Constituency parsing:

- Models natural language as a context free grammar
- Breaks text down into a phrase structure (constituents)
- Constructs the syntactic structure from constituents
- **Useful when we are interested in phrases**

Further-Watching Material

Syntactic Structure: Constituency vs Dependency

https://youtu.be/svswQnMSPDM?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Empirical Data Driven Approach to Parsing

https://youtu.be/NqM0ec6ikUc?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Grammar Transforms

https://youtu.be/J1sYddZ_fl8?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

CKY Parsing

https://youtu.be/I89KPQWylIU?list=PLoROMvodv4rOFZnDyrlW3-nI7tMLtmiJZ

Thank you for your attention!
Questions?