

# Introduction to Python and Natural Language Technologies

## 10. Syntax

Dávid Márk Nemeskey

MTA SZTAKI

April 11, 2018

# Introduction

# Introduction

The word is again from Ancient Greek: “arrangement” or “coordination”

- σύν (sýn): together
- τάξις (táxis): ordering

# Introduction

The word is again from Ancient Greek: “arrangement” or “coordination”

- σύν (sýn): together
- τάξις (táxis): ordering

It is the next step up from morphology:

- *Morphology* is the set of rules that govern the structure of *words*
- **Syntax** is the set of rules that govern the structure of **sentences**

The terms syntax and (*formal*) *grammar* are usually synonymous.

# Grammar vs Semantics

- (1) *Colorless green ideas sleep furiously.*
- (2) *\*Furiously sleep ideas green colorless.*

Chomsky (1956)

# Grammar vs Semantics

- (1) *Colorless green ideas sleep furiously.*
- (2) *\*Furiously sleep ideas green colorless.*

Chomsky (1956)

Neither of the two sentences make any sense. However,

- ❶ sentence (1) is **grammatical**;
- ❷ sentence (2) is not.

# Grammar vs Semantics

- (1) *Colorless green ideas sleep furiously.*
- (2) *\*Furiously sleep ideas green colorless.*

Chomsky (1956)

Neither of the two sentences make any sense. However,

- ❶ sentence (1) is **grammatical**;
- ❷ sentence (2) is not.

Syntax concerns itself with grammaticality; whether the sentences are meaningful or not is irrelevant.

# Grammar vs Semantics

- (1) *Colorless green ideas sleep furiously.*
- (2) *\*Furiously sleep ideas green colorless.*

Chomsky (1956)

Neither of the two sentences make any sense. However,

- ① sentence (1) is **grammatical**;
- ② sentence (2) is not.

Syntax concerns itself with grammaticality; whether the sentences are meaningful or not is irrelevant.

The study of meaning is semantics, which shall be covered in later lectures.



## ① Antiquity

- Pāṇini (around 500BC): Aṣṭādhyāyī, a sūtra of 3,959 verses, describing the grammar of Ancient Sanskrit

## ② Traditional (high school) grammar

- The European tradition before the (middle of the) 20th century
- Focused on grammatical relations (subject, object, etc.)

## ③ Phrase-structure grammar (PSG)

- PSG and formal grammars were defined by Noah Chomsky (1956)
- It became the predominant linguistic theory in the second half of the 20th century

## ④ Modern grammars

- The Chomskyan paradigm (the latest version is *Minimalism*) is still going strong
- Grammatical relations have made a comeback, e.g. in *Dependency Grammar*
- A whole landscape of various formalisms

# Concepts

We need to define a few concepts to formalize the notion of grammar

- Part of speech
- Grammatical relations
- Subcategorization
- Constituency

# Concepts

We need to define a few concepts to formalize the notion of grammar

- Part of speech
- Grammatical relations
- Subcategorization
- Constituency

In the following, we mainly concentrate on

- English syntax
- significant phenomena from other languages

# Concepts: Part of speech

A **part of speech (PoS)** is a category of words with similar grammatical distribution. Some of these are:

- Noun: *dogs, John*
- Verb: *sleep, was*
- Adjective: *blue, quicker*
- Adverb: *very, quickly*
- Preposition: *for, to*
- Determiners: *the, that*
- ...

# Concepts: Part of speech

A **part of speech (PoS)** is a category of words with similar grammatical distribution. Some of these are (with Penn Treebank POS tags):

- Noun (NN\*): *dogs* (NNS), *John* (NNP)
- Verb (VB\*): *sleep* (VBZ), *was* (VBD)
- Adjective (JJ\*): *blue* (JJ), *quicker* (JJR)
- Adverb (RB\*): *very*, *quickly*
- Preposition (IN): *for*, *to*
- Determiners (DT): *the*, *that*
- ...

# Concepts: Part of speech

A **part of speech (PoS)** is a category of words with similar grammatical distribution. Some of these are (with Penn Treebank POS tags):

- Noun (NN\*): *dogs* (NNS), *John* (NNP)
- Verb (VB\*): *sleep* (VBZ), *was* (VBD)
- Adjective (JJ\*): *blue* (JJ), *quicker* (JJR)
- Adverb (RB\*): *very*, *quickly*
- Preposition (IN): *for*, *to*
- Determiners (DT): *the*, *that*
- ...

These classes can be classified as

- **Open** (the first four): new words can be added, e.g. *laser*, *to google*
- **Closed** (the last two): the membership is fixed

# Concepts: grammatical relations

The center of a sentence is the **predicate**.

- In English, this is always a *verb*, and also called the **main verb**
- Other languages, such as Hungarian or Japanese, may have *nominal* or *adjectival predicates*

(1) The sky **is** blue

(2) Az ég **kék**  
the sky blue  
'The sky is blue'

(3) 空が 青かった  
sky.SUBJ blue.PAST  
'The sky was blue'

# Concepts: grammatical relations

The predicate may have **arguments**: expressions the complete its meaning. They are usually obligatory, i.e. the sentence would be meaningless without them.

Their type depends on the language, predicate and other factors (voice, etc.).

- (1) [SUBJ John] loves [OBJ Mary].
- (2) Rose, [SUBJ ich] erinnere [ACC mich] [DAT an Ihren Vater] liebevoll.



# Concepts: grammatical relations

The arguments that English verbs may take are listed below, with their most common semantic roles:

- Subject: the agent / theme of the predicate
- Object: the entity acted upon by the subject
  - ① Direct object: entity acted upon (accusative case)
  - ② Indirect object: affected by the action (dative case)
  - ③ Oblique object: object with a preposition (all other cases)

- (1) [SUBJ John] loves [DOBJ Mary].
- (2) [SUBJ John] gives [IOBJ Mary] [DOBJ a flower].
- (3) [SUBJ John] gave [DOBJ a flower] [OBL-OBJ to Mary].

# Concepts: grammatical relations

Note that the roles on the last slide are only **approximately** valid, and differ from verb to verb. Also they were listed with the active voice in mind.

Observe how the arguments change in the passive voice, even if the meaning of the sentence remains the same.

- (1) [SUBJ Mary] was given [DOBJ a flower] by John.
- (2) [SUBJ A flower] was given [OBL-OBJ to Mary] by John.

# Concepts: grammatical relations

A sentence may also have **adjuncts**: phrases that give extra information but are *optional*.

*Adjuncts* are similar to oblique objects, but can be omitted, while **arguments** generally cannot:

- (1) John gave a flower **to Mary** *in front of the café*.
- (2) \*John gave a flower *in front of the café*. *To whom?*
- (3) John gave a flower **to Mary**.

# Concepts: subcategorization

Also **valency**: the ability of verbs that specify (limit) the number and types of its arguments.

# Concepts: subcategorization

Also **valency**: the ability of verbs that specify (limit) the number and types of its arguments.

One form of subcategorization is **transitivity**. Verbs can be

- **Intransitive**: no object (*John sleeps.*)
- **Transitive**: only direct subject (*John loves Mary.*)
- **Ditransitive**: direct and indirect subject (*John gave Mary a flower.*)

# Concepts: subcategorization

Also **valency**: the ability of verbs that specify (limit) the number and types of its arguments.

One form of subcategorization is **transitivity**. Verbs can be

- **Intransitive**: no object (*John sleeps.*)
- **Transitive**: only direct subject (*John loves Mary.*)
- **Ditransitive**: direct and indirect subject (*John gave Mary a flower.*)

Requirements on argument types (semantics has more to say on this):

- (1) I wanted a cake.
- (2) I bought a cake.
- (3) I wanted to go.
- (4) \*I bought to go.

# Concepts: constituency

A **constituent** or **phrase** is a group of words that acts as a unit and fulfills one of the grammatical roles in the sentence.

---

<sup>1</sup>Although see [https://en.wikipedia.org/wiki/Determiner\\_phrase](https://en.wikipedia.org/wiki/Determiner_phrase)

# Concepts: constituency

A **constituent** or **phrase** is a group of words that acts as a unit and fulfills one of the grammatical roles in the sentence.

Major constituents:

- Noun phrases (NP): *John, the poor dog, the girl I saw yesterday*
- Verb phrases (VP): *slept, loves Mary*
- Prepositional phrases (PP): *from him, in front of the café*

---

<sup>1</sup>Although see [https://en.wikipedia.org/wiki/Determiner\\_phrase](https://en.wikipedia.org/wiki/Determiner_phrase)



# Concepts: constituency

A **constituent** or **phrase** is a group of words that acts as a unit and fulfills one of the grammatical roles in the sentence.

Major constituents:

- Noun phrases (NP): *John, the poor dog, the girl I saw yesterday*
- Verb phrases (VP): *slept, loves Mary*
- Prepositional phrases (PP): *from him, in front of the café*

Minor phrases (occur as parts of the major ones):

- Adjectival phrases (AP): *rather slow*
- Adverbial phrases (AdvP): *very quickly*

---

<sup>1</sup>Although see [https://en.wikipedia.org/wiki/Determiner\\_phrase](https://en.wikipedia.org/wiki/Determiner_phrase)

# Concepts: constituency

A **constituent** or **phrase** is a group of words that acts as a unit and fulfills one of the grammatical roles in the sentence.

Major constituents:

- Noun phrases (NP): *John, the poor dog, the girl I saw yesterday*
- Verb phrases (VP): *slept, loves Mary*
- Prepositional phrases (PP): *from him, in front of the café*

Minor phrases (occur as parts of the major ones):

- Adjectival phrases (AP): *rather slow*
- Adverbial phrases (AdvP): *very quickly*

Some POS tags do not have associated phrases: DET<sup>1</sup>, AUX, PUNCT.

---

<sup>1</sup>Although see [https://en.wikipedia.org/wiki/Determiner\\_phrase](https://en.wikipedia.org/wiki/Determiner_phrase)

# Concepts: constituency test

Several tests exist to check if a group of words is a constituent. The first is the *substitution test*: noun phrases, but not their parts, can be substituted for one another:

- (1) **The poor dog** bit **John**.
- (2) **John** bit **the poor dog**.
- (3) \*The John bit poor dog.

# Concepts: constituency test

Several tests exist to check if a group of words is a constituent. The first is the *substitution test*: noun phrases, but not their parts, can be substituted for one another:

- (1) **The poor dog** bit **John**.
- (2) **John** bit **the poor dog**.
- (3) \*The John bit poor dog.

Another example is the *topicalization test*: prepositional phrases can be *topicalized*, but only whole:

- (1) John gave Mary a flower **in front of the café**.
- (2) **In front of the café**, John gave Mary a flower.
- (3) \*In front, John gave Mary a flower of the café.

# Concepts: constituency test

Several tests exist to check if a group of words is a constituent. The first is the *substitution test*: noun phrases, but not their parts, can be substituted for one another:

- (1) **The poor dog** bit **John**.
- (2) **John** bit **the poor dog**.
- (3) \*The John bit poor dog.

Another example is the *topicalization test*: prepositional phrases can be *topicalized*, but only whole:

- (1) John gave Mary a flower **in front of the café**.
- (2) **In front of the café**, John gave Mary a flower.
- (3) \*In front, John gave Mary a flower of the café.

These tests also present arguments for the existence of constituents.

# Phrase-structure Grammar

# Phrase-structure Grammar

The most widely known formal grammar for English is the **Phrase-structure Grammar (PSG)**.

The main units it deals with are *phrases* or *constituents*. Traditional grammatical functions are not marked explicitly.

# Phrase-structure Grammar

The most widely known formal grammar for English is the **Phrase-structure Grammar (PSG)**.

The main units it deals with are *phrases* or *constituents*. Traditional grammatical functions are not marked explicitly.

It is a **generative grammar**:

- it consists of a set of **production rules**
- these rules can be used to generate a number of *sentences*
- The **language**  $\mathcal{L}$  defined by the grammar is the set of sentences it can generate.



# PSG Components

## ① Production (replacement) rules:

NP	→	ProperNoun
NP	→	Det Nominal
Nominal	→	Noun   Nominal Noun

# PSG Components

## ① Production (replacement) rules:

NP	→	ProperNoun
NP	→	Det Nominal
Nominal	→	Noun   Nominal Noun

## ② Lexicon:

Det	→	<i>the   a   an</i>
ProperNoun	→	<i>John   Mary   ...</i>
Noun	→	<i>dog   cat   ...</i>

# PSG Components

## ① Production (replacement) rules:

NP	→	ProperNoun
NP	→	Det Nominal
Nominal	→	Noun   Nominal Noun

## ② Lexicon:

Det	→	<i>the   a   an</i>
ProperNoun	→	<i>John   Mary   ...</i>
Noun	→	<i>dog   cat   ...</i>

There are two types of symbols:

- **terminals** (*and*, *John*) are part of the generated language
- **nonterminals** (Nominal) represent word classes or constituents

# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

Example: NP

# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

Example:  $NP \rightarrow Det \text{ Nominal}$

# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

Example:  $NP \rightarrow \text{Det Nominal} \rightarrow a \text{ Nominal}$

# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

Example:  $NP \rightarrow Det \ Nominal \rightarrow a \ Nominal \rightarrow a \ Noun$



# Derivation

Text generation works as follows:

- We start with a string of a single nonterminal, the **start symbol** (usually  $S$ )
- At each step, we apply a rule:
  - We pick a nonterminal from the string
  - Choose one of the rules where that nonterminal is on the left hand side
  - Replace it in the string with the right hand side of the rule.
- The process ends when the string consists entirely of terminal symbols.

The sequence of rule applications is called the **derivation** of the final string.

Example:  $NP \rightarrow Det \ Nominal \rightarrow a \ Nominal \rightarrow a \ Noun \rightarrow a \ dog$

# The parse tree

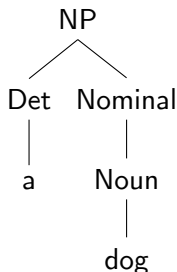
A **Parse tree** or **syntax tree** provides a graphical view on the derivation process:

- An inverted tree
- Inner nodes are nonterminal symbols
- The leaves are terminals
- Edges signify rule applications

# The parse tree

A **Parse tree** or **syntax tree** provides a graphical view on the derivation process:

- An inverted tree
- Inner nodes are nonterminal symbols
- The leaves are terminals
- Edges signify rule applications



# Example

This is our example grammar:

S	→ NP VP	<i>Subject-predicate</i>
NP	→ Pronoun   ProperNoun   Det Nominal	<i>One way to list alternatives</i> <i>Another way</i>
Nominal	→ Nominal Noun   Noun	
VP	→ Verb   Verb NP   Verb PP   Verb NP PP	<i>Intransitive verb</i> <i>Transitive verb</i> <i>Oblique / adjunct</i> <i>Oblique / adjunct</i>
PP	→ Preposition NP	

# Example

Rule to apply:

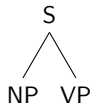
String: S

S

# Example

Rule to apply:  $S \rightarrow NP \ VP$

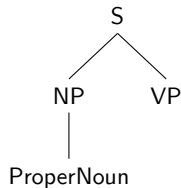
String: NP VP



# Example

Rule to apply:  $NP \rightarrow \text{ProperNoun}$

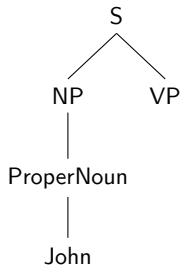
String: ProperNoun VP



# Example

Rule to apply: ProperNoun  $\rightarrow$  *John*

String: *John VP*

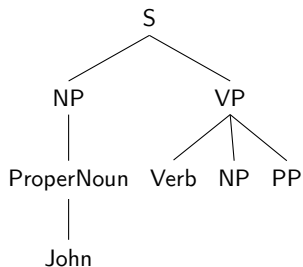




# Example

Rule to apply:  $VP \rightarrow \text{Verb NP PP}$

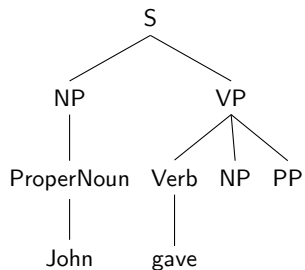
String: *John* Verb NP PP



# Example

Rule to apply: Verb  $\rightarrow$  *gave*

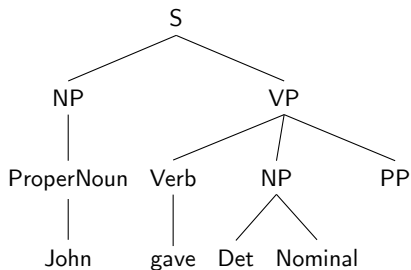
String: *John gave* NP PP



# Example

Rule to apply:  $\text{NP} \rightarrow \text{Det Nominal}$

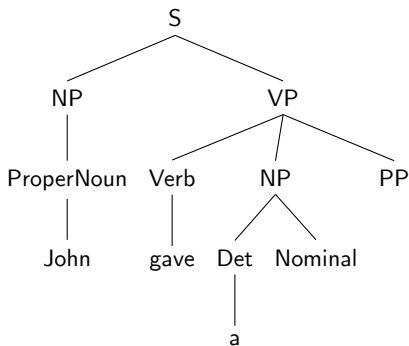
String: *John gave* Det Nominal PP



# Example

Rule to apply:  $\text{Det} \rightarrow a$

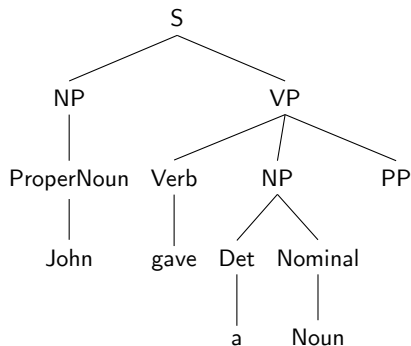
String: *John gave a Nominal PP*



# Example

Rule to apply: Nominal  $\rightarrow$  Noun

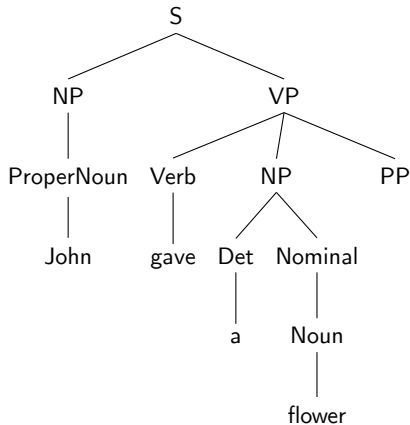
String: *John gave a Noun PP*



# Example

Rule to apply: Noun  $\rightarrow$  *flower*

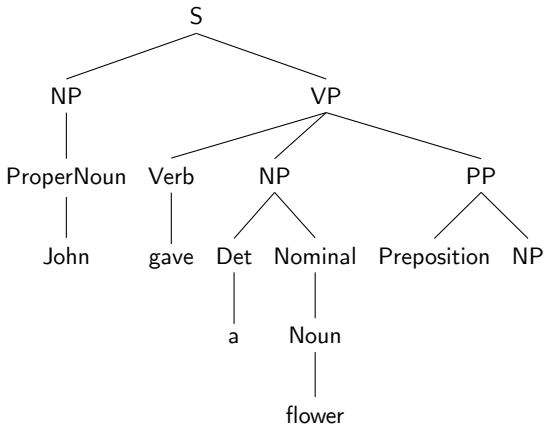
String: *John gave a flower* PP



# Example

Rule to apply:  $PP \rightarrow \text{Preposition NP}$

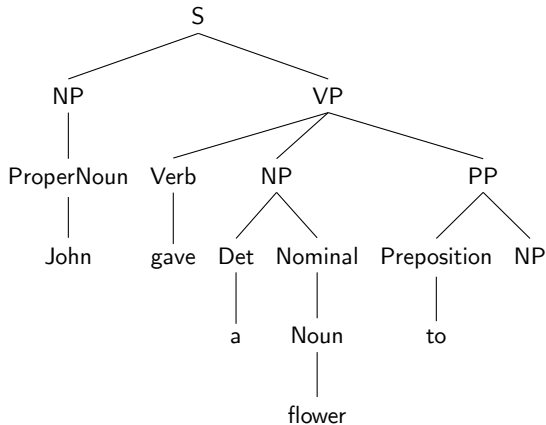
String: *John gave a flower* Preposition NP



# Example

Rule to apply: Preposition  $\rightarrow$  to

String: *John gave a flower to* NP

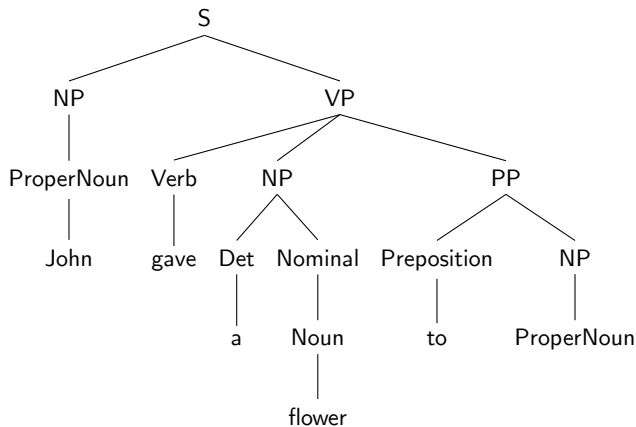




# Example

Rule to apply:  $NP \rightarrow \text{ProperNoun}$

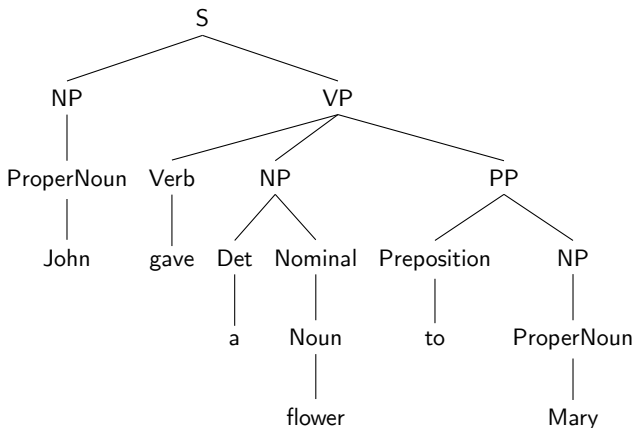
String: *John gave a flower to* ProperNoun



# Example

Rule to apply: ProperNoun  $\rightarrow$  *Mary*

String: *John gave a flower to Mary*



# Bracketed notation

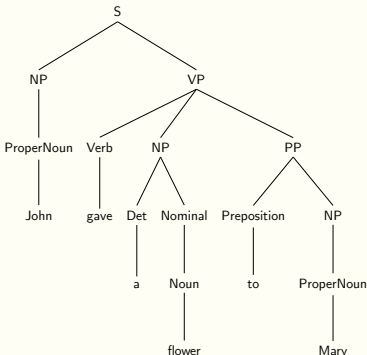
The **bracketed notation** is a way to represent the parse tree in text. This format is used by software (such as the online version of the Stanford Parser) to process trees.

# Bracketed notation

The **bracketed notation** is a way to represent the parse tree in text. This format is used by software (such as the online version of the Stanford Parser) to process trees.

Example:

```
(S
  (NP
    (ProperNoun John))
  (VP
    (Verb gave)
    (NP
      (Det a)
      (Nominal
        (Noun flower)))
    (PP
      (TO to)
      (NP
        (ProperNoun Mary)))))
```



# The Chomsky Normal Form

A grammar is in the **Chomsky Normal Form (CNF)**, if all its rules take one of two forms:

$$A \rightarrow B C$$
$$A \rightarrow a$$

# The Chomsky Normal Form

A grammar is in the **Chomsky Normal Form (CNF)**, if all its rules take one of two forms:

$$A \rightarrow B C$$

$$A \rightarrow a$$

CNF has two interesting properties:

- CNF grammars produce binary parsing trees
- Every PSG grammar can be converted into CNF

# The Chomsky Normal Form

A grammar is in the **Chomsky Normal Form (CNF)**, if all its rules take one of two forms:

$$A \rightarrow B C$$

$$A \rightarrow a$$

CNF has two interesting properties:

- CNF grammars produce binary parsing trees
- Every PSG grammar can be converted into CNF

Some algorithms only work on CNF grammars.

# Parsing



# What can we do with grammars?

- Obtain a grammar
  - The lexicon can be trivially derived from dictionaries or morphological lexicons
  - Production rules are usually written by linguists (but see next section...)

# What can we do with grammars?

- Obtain a grammar
  - The lexicon can be trivially derived from dictionaries or morphological lexicons
  - Production rules are usually written by linguists (but see next section...)
- Parse a text
  - Given a text, return the syntax tree (the “**parse**”)
  - Enumerating all possible parses is not difficult
  - Picking the *most correct* parse is

# What can we do with grammars?

- Obtain a grammar
  - The lexicon can be trivially derived from dictionaries or morphological lexicons
  - Production rules are usually written by linguists (but see next section...)
- Parse a text
  - Given a text, return the syntax tree (the “**parse**”)
  - Enumerating all possible parses is not difficult
  - Picking the *most correct* parse is
- Generate a text
  - The “inverse” of parsing (see previous example)
  - We are not going into details in this course

# Parsing

Input: The text sentence

Output: A parse tree

# Parsing

**Input:** The text sentence

**Output:** A parse tree

There are two directions we can build the tree from:

## ① Top-down

- We start from the root nonterminal, S
- Apply rules until we reach the terminals (as if we were generating)
- Parsing finishes when the whole sentence is generated

# Parsing

**Input:** The text sentence

**Output:** A parse tree

There are two directions we can build the tree from:

## ① Top-down

- We start from the root nonterminal,  $S$
- Apply rules until we reach the terminals (as if we were generating)
- Parsing finishes when the whole sentence is generated

## ② Bottom-up

- Start from the sentence
- Apply rules backwards: if we have a sub-graph that matches the right-hand side of a rule, add the left-hand side node, and connect it with the right-hand side
- Parsing finishes if we reach  $S$  and have a single tree

# Challenges

There are two main challenges that parsing algorithms must face:

- **Nondeterminism** of the grammar
- **Ambiguity** of the natural language

# Challenge #1: Nondeterminism

Natural language grammars are nondeterministic:

- Each node could be expanded in several ways
- Examples:
  - $VP \rightarrow \text{Verb} \mid \text{Verb NP} \mid \text{Verb PP} \mid \text{Verb NP PP}$
  - $\text{Verb} \rightarrow \text{give} \mid \text{take} \mid \text{see} \mid \dots$
- There is only one (or at most few) correct choice for a particular sentence



# Challenge #1: Nondeterminism

Natural language grammars are nondeterministic:

- Each node could be expanded in several ways
- Examples:
  - $VP \rightarrow \text{Verb} \mid \text{Verb NP} \mid \text{Verb PP} \mid \text{Verb NP PP}$
  - $\text{Verb} \rightarrow \text{give} \mid \text{take} \mid \text{see} \mid \dots$
- There is only one (or at most few) correct choice for a particular sentence

Building the tree can be rephrased into a graph search problem:

- Each node in the search graph represents a partially completed parse tree
- Nodes are visited according to a **search strategy**: BFS, DFS,  $A^*$ , ...
- When the node is not compatible with the sentence, the algorithm **backtracks**

# Top-down parse example

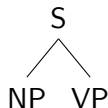
Let's parse "*John loves Mary.*" with our toy grammar.

S

# Top-down parse example

Let's parse "*John loves Mary.*" with our toy grammar.

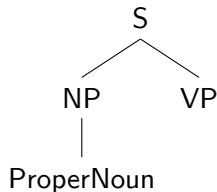
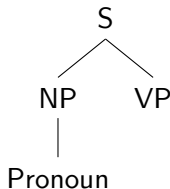
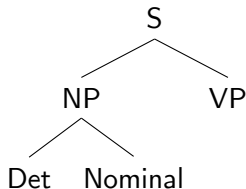
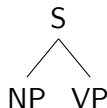
S



# Top-down parse example

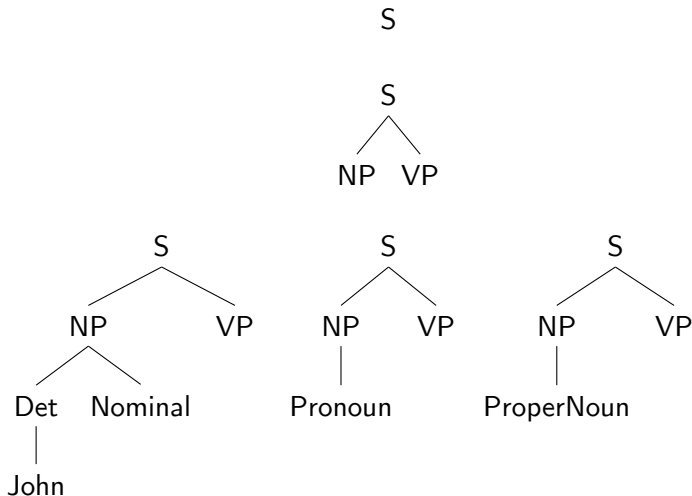
Let's parse "*John loves Mary.*" with our toy grammar.

S



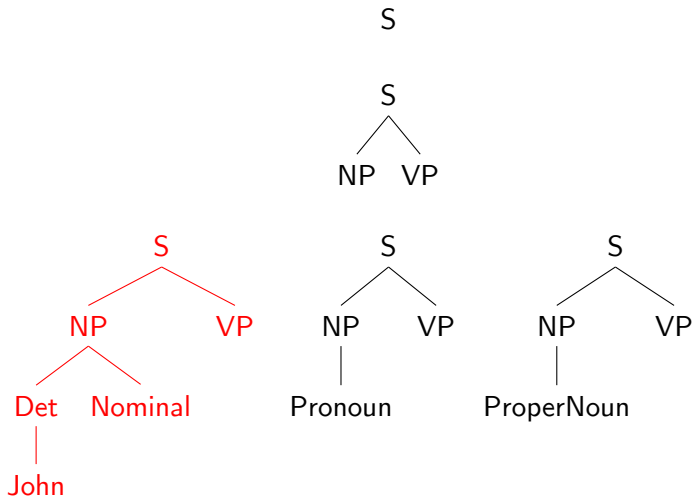
# Top-down parse example

Let's parse "*John loves Mary.*" with our toy grammar.



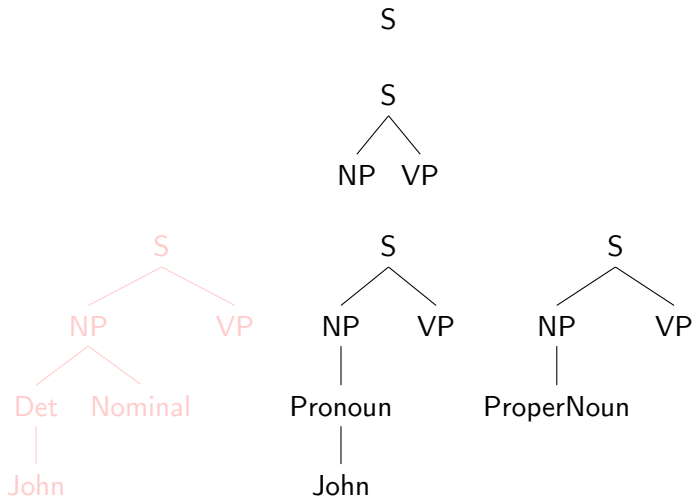
# Top-down parse example

Let's parse "*John loves Mary.*" with our toy grammar.



# Top-down parse example

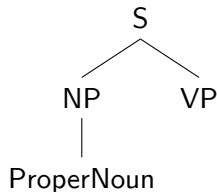
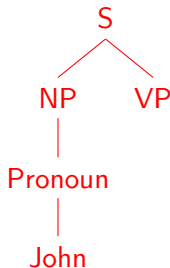
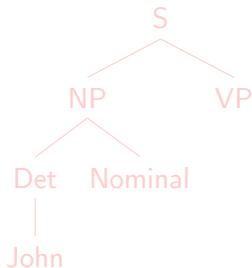
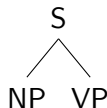
Let's parse "*John loves Mary.*" with our toy grammar.



# Top-down parse example

Let's parse "*John loves Mary.*" with our toy grammar.

S

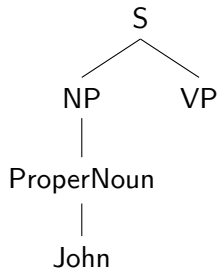
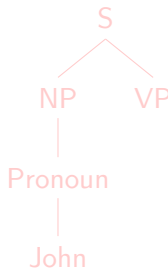
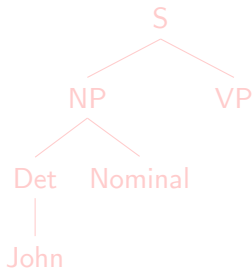
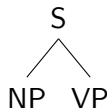




# Top-down parse example

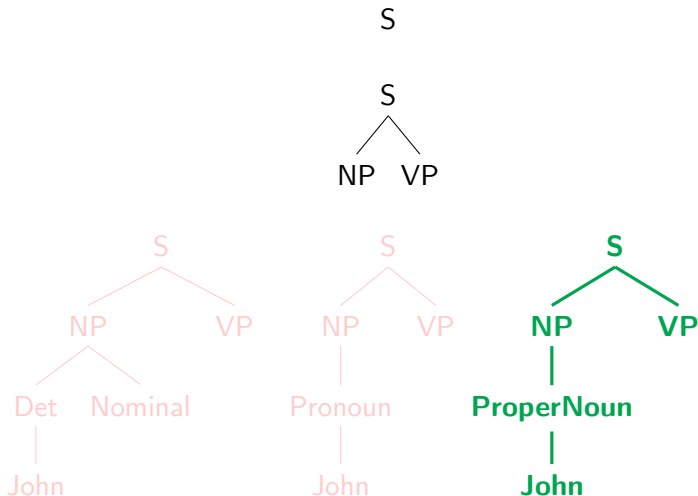
Let's parse "*John loves Mary.*" with our toy grammar.

S



# Top-down parse example

Let's parse "*John loves Mary.*" with our toy grammar.



# Top-down parse example – cont.

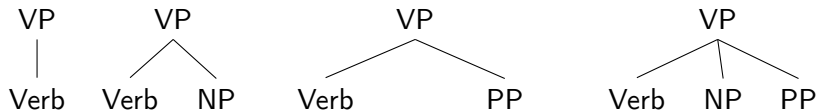
Let's parse "... *loves Mary.*"

VP

# Top-down parse example – cont.

Let's parse "... loves Mary."

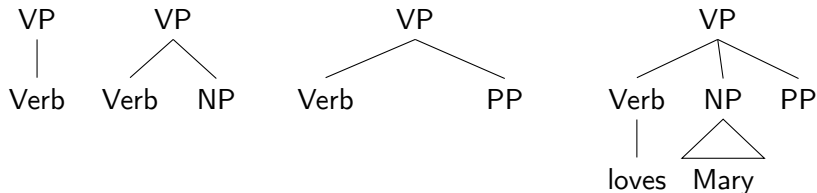
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

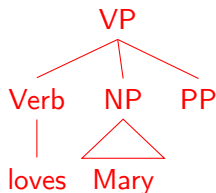
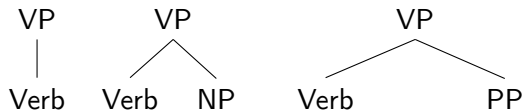
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

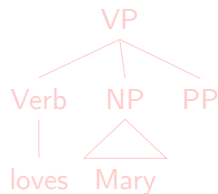
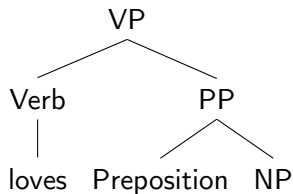
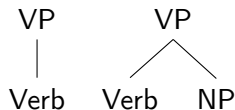
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

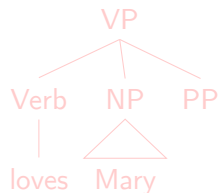
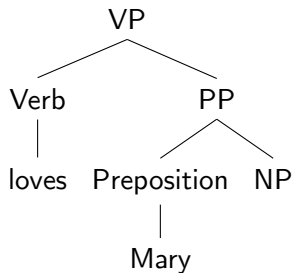
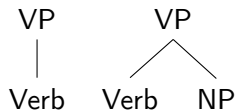
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

VP

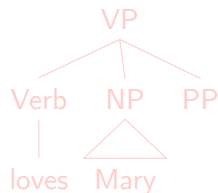
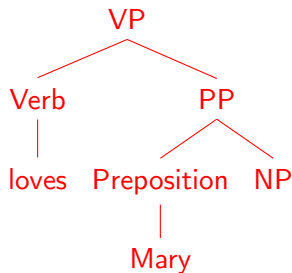
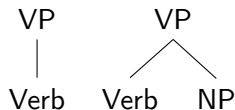




# Top-down parse example – cont.

Let's parse "... loves Mary."

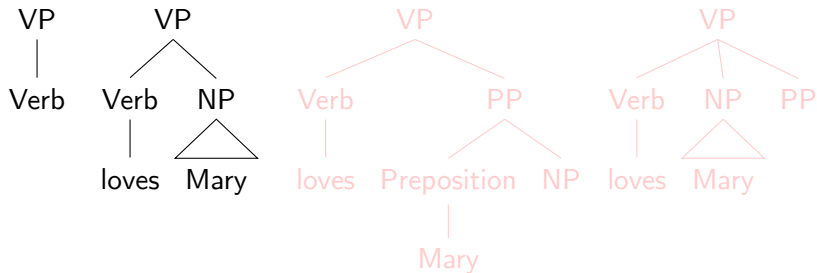
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

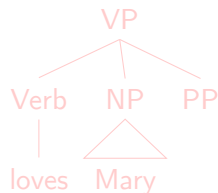
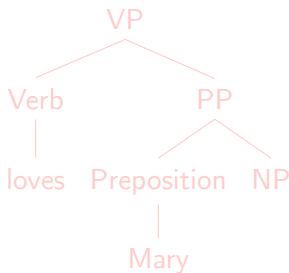
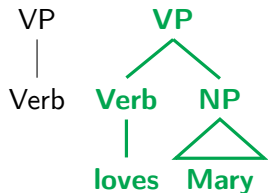
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

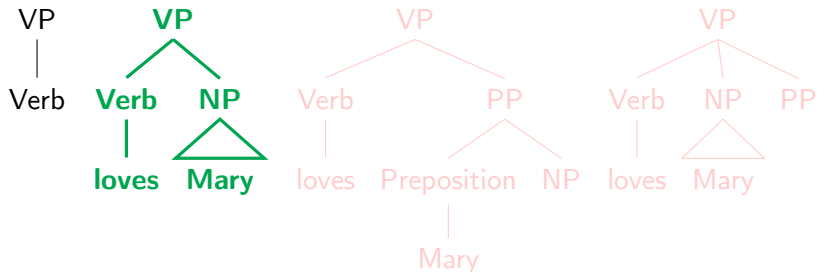
VP



# Top-down parse example – cont.

Let's parse "... loves Mary."

VP



Even parsing a simple sentence with a toy grammar,

- top-down parsing backtracks a lot (bottom-up would, too)
- the same sub-trees are built and thrown away multiple times ([Mary]<sub>NP</sub>)

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an elephant in my pajamas.*

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

- The PP “*in my pajamas*” can attach to the VP and the NP
- Disambiguation requires semantic and pragmatic knowledge – not available at this point in the usual linguistic pipeline



## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I **shot** an elephant **in my pajamas**. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

- The PP “*in my pajamas*” can attach to the **VP** and the NP
- Disambiguation requires semantic and pragmatic knowledge – not available at this point in the usual linguistic pipeline

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an **elephant in my pajamas**. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

- The PP “*in my pajamas*” can attach to the VP and the **NP**
- Disambiguation requires semantic and pragmatic knowledge – not available at this point in the usual linguistic pipeline

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

- The PP “*in my pajamas*” can attach to the VP and the NP
- Disambiguation requires semantic and pragmatic knowledge – not available at this point in the usual linguistic pipeline

## Challenge #2: Ambiguity

A sentence is **ambiguous**, if it has more than one possible (correct) parse.

*One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*

*Groucho Marx, Animal Crackers, 1930*

- The PP “*in my pajamas*” can attach to the VP and the NP
- Disambiguation requires semantic and pragmatic knowledge – not available at this point in the usual linguistic pipeline

*John saw the man on the mountain with a telescope.*

- This sentence is ambiguous even semantically

# Consequences of ambiguity

There are two forms of ambiguity:

- **Global ambiguity:** the whole sentence is ambiguous; it has multiple parses.
- **Local ambiguity:** a constituent is ambiguous in itself, but not when part of the sentence. This induces backtracking in the parser.

# Consequences of ambiguity

There are two forms of ambiguity:

- **Global ambiguity:** the whole sentence is ambiguous; it has multiple parses.
- **Local ambiguity:** a constituent is ambiguous in itself, but not when part of the sentence. This induces backtracking in the parser.

Some types of ambiguity:

- Attachment ambiguity: it is not clear where the constituent should attach. *I shot an elephant in my pajamas.*
- Coordination ambiguity: it is ambiguous what phrases a conjunction connects. *I saw old men and women.*
- Homonymy: words with different meanings or POS tag can take the same form. E.g. *loves* can be both a noun and a verb.

# Dynamic programming

We need an algorithm that

- Takes a sentence and returns “the correct” parse tree(s)
- Does not backtrack much
- Does not duplicate work by repeatedly building the same subtree

# Dynamic programming

We need an algorithm that

- Takes a sentence and returns “the correct” parse tree(s)
- Does not backtrack much
- Does not duplicate work by repeatedly building the same subtree

**Dynamic programming** is a group of algorithms that

- breaks the problem into smaller subproblems
- solves each subproblem only once
- stores the solutions to (re)use it later



# Dynamic programming

We need an algorithm that

- Takes a sentence and returns “the correct” parse tree(s)
- Does not backtrack much
- Does not duplicate work by repeatedly building the same subtree

**Dynamic programming** is a group of algorithms that

- breaks the problem into smaller subproblems
- solves each subproblem only once
- stores the solutions to (re)use it later

In our case, these subproblems are the subtrees of the whole syntax tree.

# The CKY algorithm

The **CKY** (short for **Cocke-Kasami-Younger**) algorithm is a bottom-up parsing algorithm:

- dynamic programming
- requires that the grammar is in CNF
- runs in  $\mathcal{O}(n^3)$  time.

# The CKY algorithm

The **CKY** (short for **Cocke-Kasami-Younger**) algorithm is a bottom-up parsing algorithm:

- dynamic programming
- requires that the grammar is in CNF
- runs in  $\mathcal{O}(n^3)$  time.

In a nutshell:

- For a sentence of  $n$  words, CKY fills the upper triangle of an  $n^2$  matrix
- Each cell corresponds to a number of consecutive words
- The cells record all possible constituents that cover those words
- A parse is successful, if the top right cell contains an S

# Example

In this example, we are going to parse the sentence *The dog bit John*.

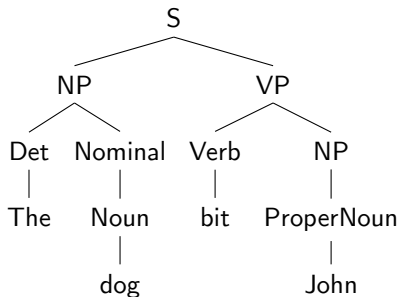
- 1 convert the grammar to CNF
- 2 run the CKY algorithm.

# Example

In this example, we are going to parse the sentence *The dog bit John.*

- 1 convert the grammar to CNF
- 2 run the CKY algorithm.

The parse we are looking for is:



# Example: conversion to CNF

Reminder: the CNF contains two types of rules:

$$A \rightarrow B \ C$$

$$A \rightarrow a$$

# Example: conversion to CNF

Reminder: the CNF contains two types of rules:

$$A \rightarrow B C$$

$$A \rightarrow a$$

In Phrase Structure Grammars, lexical rules already conform to the second type.

Production rules that do not conform to the first type belong to two groups:

① Right-hand side too long:  $A \rightarrow B C D$

② Unit productions:  $A \rightarrow B$

# Example: conversion to CNF

Rules whose right-hand side is too long:

$$VP \rightarrow \text{Verb NP PP}$$



# Example: conversion to CNF

Rules whose right-hand side is too long:

$$VP \rightarrow \text{Verb NP PP}$$

are split into two by introducing a new nonterminal:

$$VP \rightarrow \text{Verb VP'}$$
$$VP' \rightarrow \text{NP PP}$$

This step can be applied recursively, if needed.

# Example: conversion to CNF

Unit productions:

S  $\rightarrow$  NP VP

NP  $\rightarrow$  ProperNoun

**Unit production**

# Example: conversion to CNF

Unit productions:

S  $\rightarrow$  NP VP

NP  $\rightarrow$  ProperNoun

**Unit production**

Unit production rules are deleted, their right-hand side is “pulled up” to all rules that contain the left-hand nonterminal on their right.

S  $\rightarrow$  NP VP

S  $\rightarrow$  ProperNoun VP

# Example: conversion of our toy grammar

S	→	NP VP	<i>Subject-predicate</i>
NP	→	Pronoun   ProperNoun   Det Nominal	
Nominal	→	Nominal Noun   Noun	
VP	→	Verb   Verb NP   Verb PP   Verb NP PP	<i>Intransitive verb</i> <i>Transitive verb</i> <i>Oblique / adjunct</i> <i>Oblique / adjunct</i>
PP	→	Preposition NP	

## Example: conversion of our toy grammar

S → NP VP                      *Subject-predicate*

```
NP      → ProperNoun
        | Det Nominal
```

$$\begin{array}{ccc} \text{Nominal} & \rightarrow & \text{Nominal Noun} \\ & | & \text{Noun} \end{array}$$

VP	→	Verb	<i>Intransitive verb</i>
		Verb NP	<i>Transitive verb</i>

The subset of rules needed for the example (*The dog bit John*)

# Example: conversion of our toy grammar

S → NP VP

NP → ProperNoun                      **Unit production**  
    | Det Nominal

Nominal → Nominal Noun  
          | Noun                      **Unit production**

VP → Verb                      **Unit production**  
    | Verb NP

Let's get rid of the rest of unit productions,  $NP \rightarrow \text{ProperNoun}$  first.

# Example: conversion of our toy grammar

S           → NP VP  
          | ProperNoun VP

NP           → Det Nominal

Nominal   → Nominal Noun  
          | Noun

**Unit production**

VP           → Verb  
          | Verb NP  
          | Verb ProperNoun

**Unit production**

... then Nominal → Noun ...

# Example: conversion of our toy grammar

S           → NP VP  
          | ProperNoun VP

NP           → Det Nominal  
          | Det Noun

Nominal     → Nominal Noun  
          | Noun Noun

VP           → Verb  
          | Verb NP  
          | Verb ProperNoun

**Unit production**

... and finally  $VP \rightarrow Verb$ .



# Example: conversion of our toy grammar

S           → NP VP  
          | ProperNoun VP  
          | NP Verb  
          | ProperNoun Verb

NP           → Det Nominal  
          | Det Noun

Nominal     → Nominal Noun  
          | Noun Noun

VP           → Verb NP  
          | Verb ProperNoun

The CNF form of our grammar.

## Example: conversion of our toy grammar

S           → NP VP  
          | ProperNoun VP  
          | NP Verb  
          | ProperNoun Verb

NP          → Det Nominal  
          | Det Noun

Nominal    → Nominal Noun  
          | Noun Noun

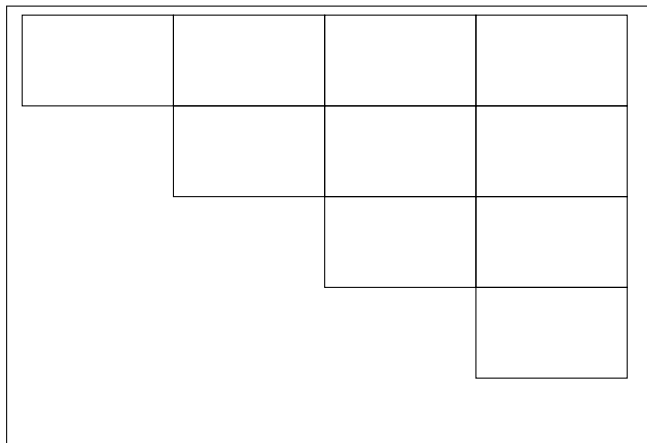
VP          → Verb NP  
          | Verb ProperNoun

The CNF form of our grammar. Note how we started out with 7 rules and ended up with 10: conversion to CNF can lead to a proliferation of rules.

# Example: CKY – The dog bit John.

Now we can apply the CKY algorithm...

## Example: CKY – The dog bit John.



First, we start by drawing the upper triangular matrix

## Example: CKY – The dog bit John.

<i>the</i>			
	<i>dog</i>		
		<i>bit</i>	
			<i>John</i>

Each column is assigned to a word in the sentence

## Example: CKY – The dog bit John.

[1]	[1-2]	[1-3]	[1-4]
<i>the</i>	[2]	[2-3]	[2-4]
	<i>dog</i>	[3]	[3-4]
		<i>bit</i>	[4]
			<i>John</i>

We note the word indices each cell corresponds to

## Example: CKY – The dog bit John.

[1] <b>Det</b>	[1-2]	[1-3]	[1-4]
<i>the</i>	[2] <b>Noun</b>	[2-3]	[2-4]
	<i>dog</i>	[3] <b>Noun Verb</b>	[3-4]
		<i>bit</i>	[4] <b>ProperNoun</b>
			<i>John</i>

The diagonal records the terminal production rules (POS). Ambiguity: *bit* can also be a noun

## Example: CKY – The dog bit John.

[1] Det <i>the</i>	[1-2]	[1-3]	[1-4]
	[2] Noun <i>dog</i>	[2-3]	[2-4]
		[3] Noun Verb <i>bit</i>	[3-4]
			[4] ProperNoun <i>John</i>

There is not much else to do in the first column



## Example: CKY – The dog bit John.

[1]	Det	[1-2]	[1-3]	[1-4]
<i>the</i>		Noun	[2-3]	[2-4]
		<i>dog</i>	Noun Verb	[3-4]
			<i>bit</i>	ProperNoun [4]
				<i>John</i>

On to the second column...

## Example: CKY – The dog bit John.

[1]	Det [1-2]	[1-3]	[1-4]
<i>the</i>	Noun [2]	[2-3]	[2-4]
	<i>dog</i>	Noun Verb [3]	[3-4]
		<i>bit</i>	ProperNoun [4]
			<i>John</i>

Non-leaf cells correspond to binary rules:

- the first constituent on the right side of the rule is to the *left*
- the second one is *down*

## Example: CKY – The dog bit John.

[1]	Det [1-2]	[1-3]	[1-4]
<i>the</i>	Noun [2]	[2-3]	[2-4]
	<i>dog</i>	Noun Verb [3]	[3-4]
		<i>bit</i>	ProperNoun [4]
			<i>John</i>

The first rule application:  $NP \rightarrow Det \ Noun$

## Example: CKY – The dog bit John.

[1] Det	[1-2] NP	[1-3]	[1-4]
<i>the</i>	[2] Noun	[2-3]	[2-4]
	<i>dog</i>	[3] Noun Verb	[3-4]
		<i>bit</i>	ProperNoun [4]
			<i>John</i>

On to the third column...

# Example: CKY – The dog bit John.

[1]	Det	NP	[1-3]	[1-4]
<i>the</i>		Noun	<b>← Nominal</b>	
	[2]		[2-3]	[2-4]
	<i>dog</i>		Noun	
		[3]	Verb	[3-4]
		<i>bit</i>		ProperNoun
			[4]	
				<i>John</i>

Rule application: Nominal  $\rightarrow$  Noun Noun

## Example: CKY – The dog bit John.

[1] Det	[1-2] NP	[1-3]	[1-4]
<i>the</i>	[2] Noun	[2-3] Nominal	[2-4]
	<i>dog</i>	[3] Noun Verb	[3-4]
		<i>bit</i>	[4] ProperNoun
			<i>John</i>

In general, non-leaf cells with  $N$  words can be split into two in  $N - 1$  ways.

## Example: CKY – The dog bit John.

[1]	Det	← NP	NP
	[1-2]	[1-3]	[1-4]
<i>the</i>	Noun	Nominal	
	[2]	[2-3]	[2-4]
<i>dog</i>	Noun	Verb	
	[3]	[3-4]	
<i>bit</i>	ProperNoun		
	[4]		
<i>John</i>			

In general, non-leaf cells with  $N$  words can be split into two in  $N - 1$  ways. When  $N = 3$ :

- 2 to the left, 1 down: rule application  $NP \rightarrow Det \text{ Nominal}$

## Example: CKY – The dog bit John.

[1]	Det	NP	NP	[1-4]
	[1-2]	[1-3]	S	
<i>the</i>	Noun	Nominal		[2-4]
	[2]	[2-3]		
<i>dog</i>		Noun		[3-4]
		[3]	Verb	
<i>bit</i>				
			ProperNoun	[4]
			<i>John</i>	

In general, non-leaf cells with  $N$  words can be split into two in  $N - 1$  ways. When  $N = 3$ :

- 2 to the left, 1 down: rule application  $NP \rightarrow Det \text{ Nominal}$
- 1 to the left, 2 down: rule application  $S \rightarrow NP \text{ Verb}$



## Example: CKY – The dog bit John.

[1]	Det	NP	NP S	[1-4]
<i>the</i>	[2]	Noun	Nominal	[2-4]
	<i>dog</i>	[3]	Noun Verb	[3-4]
		<i>bit</i>	ProperNoun	[4]
			<i>John</i>	

Here we found an S. However, it is not at the top right cell, so we are not done yet.

Example: CKY – The dog bit John.

[1]	Det	NP	NP S	[1-4]
	<i>the</i>	Noun	Nominal	[2-4]
		[2]		
		<i>dog</i>	Noun Verb	[3-4]
			[3]	
			<i>bit</i>	ProperNoun [4]
				<i>John</i>

On to the fourth column...

# Example: CKY – The dog bit John.

[1]	Det	[1-2]	NP	[1-3]	NP S	[1-4]
<i>the</i>			Noun	[2-3]	Nominal	[2-4]
		<i>dog</i>			Noun Verb	[3-4]
				<i>bit</i>		ProperNoun [4]
						<i>John</i>

Rule application:  $VP \rightarrow \text{Verb ProperNoun}$

# Example: CKY – The dog bit John.

[1]	Det	[1-2]	NP	[1-3]	NP S	[1-4]
<i>the</i>			Noun	← Nominal		
		[2]		[2-3]		[2-4]
		<i>dog</i>		Noun Verb		VP
			[3]		[3-4]	
			<i>bit</i>			ProperNoun
					[4]	
						<i>John</i>

No applicable rules

# Example: CKY – The dog bit John.

[1]	Det	[1-2]	NP	[1-3]	NP S	[1-4]
<i>the</i>			Noun	Nominal	←	
		[2]		[2-3]		[2-4]
		<i>dog</i>		Noun Verb		VP
			[3]		[3-4]	
			<i>bit</i>			ProperNoun
						[4]
						<i>John</i>

No applicable rules

## Example: CKY – The dog bit John.

	Det	NP	NP	
[1]	[1-2]	[1-3]	S	[1-4]
<i>the</i>	Noun	Nominal		
	[2]	[2-3]	[2-4]	
	<i>dog</i>	Noun	VP	
		[3]	[3-4]	
		<i>bit</i>	ProperNoun	
			[4]	
			<i>John</i>	

The top right cell represents the whole sentence.

# Example: CKY – The dog bit John.

	Det	NP	NP	S
[1]	[1-2]	[1-3]	[1-4]	
<i>the</i>	Noun	Nominal		
	[2]	[2-3]	[2-4]	
	<i>dog</i>	Noun		VP
		[3]	[3-4]	
		<i>bit</i>	ProperNoun	
			[4]	
			<i>John</i>	

Rule application:  $S \rightarrow NP \ VP$

# Example: CKY – The dog bit John.

[1]	Det	[1-2]	NP	[1-3]	NP S	[1-4]	S
<i>the</i>		[2]	Noun	[2-3]	Nominal	[2-4]	
	<i>dog</i>			[3]	Noun Verb	[3-4]	VP
			<i>bit</i>				ProperNoun
						[4]	
							<i>John</i>

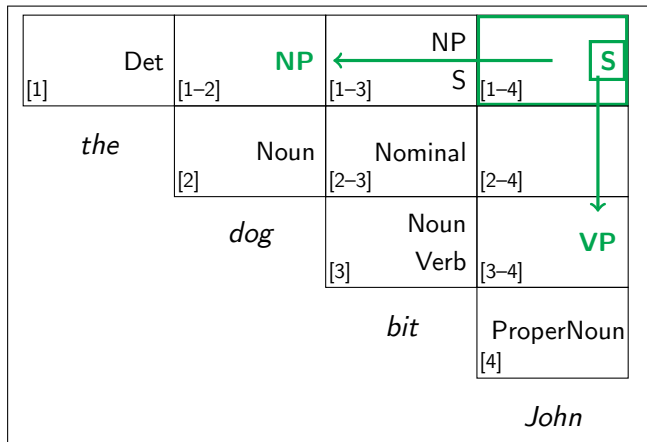


## Example: CKY – The dog bit John.

[1] Det	[1-2] NP	[1-3] NP S	[1-4] S
<i>the</i>	[2] Noun	[2-3] Nominal	[2-4]
	<i>dog</i>	[3] Noun Verb	[3-4] VP
		<i>bit</i>	[4] ProperNoun
			<i>John</i>

S in the top right cell: sentence accepted.

## Example: CKY – The dog bit John.



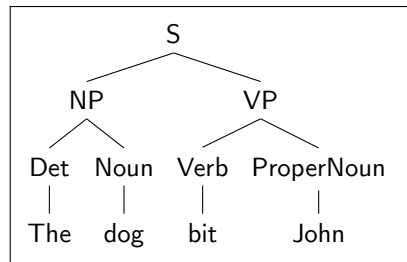
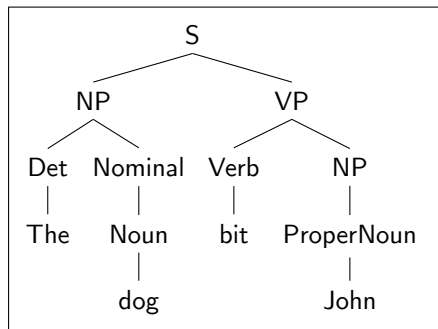
Each nonterminal maintains backpointers to its children (here: NP and VP)

## Example: CKY – The dog bit John.

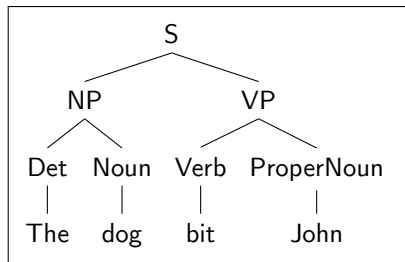
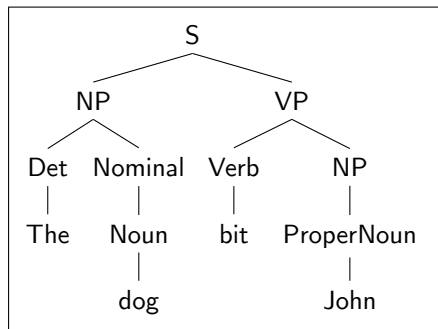
[1] <b>Det</b>	← <b>NP</b> [1-2]	← <b>NP</b> [1-3] <b>S</b>	<b>S</b> [1-4]
<i>the</i>	<b>Noun</b> [2]	<b>Nominal</b> [2-3]	[2-4]
	<i>dog</i>	<b>Noun</b> <b>Verb</b> [3]	← <b>VP</b> [3-4]
		<i>bit</i>	<b>ProperNoun</b> [4]
			<i>John</i>

The backpointers define the syntax tree.

# Example: the final tree



# Example: the final tree



Due to the CNF requirement, the CKY algorithm did not return the tree we were expecting. There are two solutions to this problem:

- convert the tree back from the CNF to the full grammar
- start from an already binary grammar

# Probabilistic PSG

# Probabilistic Parsing

The parse methods introduced previously might return multiple parses due to ambiguity.

- Not all parses are born equal
- Difficult to decide which tree is the most correct

**Probabilistic Context-Free Grammars (PCFG)** aim to solve this problem by

- assigning a probability to each production rule
- probability of a parse tree is the product of the rule probabilities:

$$P(Tree) = \prod_{i=1}^n P(rule_i)$$

- the “correct” parse is the most probable tree:  $\operatorname{argmax}_i P(Tree_i|S)$

The rule probabilities are obtained from a **treebank**:

- A text corpus annotated by linguists
- Contains the syntax tree for each sentence
- Examples:
  - English: Penn Treebank
  - Hungarian: Szeged Treebank

The probability for a rule is proportional to its normalized corpus frequency. Given a rule, e.g.  $S \rightarrow NP \ VP$ ,

$$P(S \rightarrow NP \ VP) = P(S \rightarrow NP \ VP | S) = \frac{\# [S \rightarrow NP \ VP]}{\# S}$$

Normalization by the left hand side of the rule is important, so that

$$\sum_{\alpha} P(S \rightarrow \alpha) = 1$$

, i.e. the expansions of a nonterminal form a distribution.



# PCFG: parsing and evaluation

Parsing is done with the probabilistic version of the parsing algorithms, e.g. PCKY.

Evaluation:

- On the “test” split of Treebanks
- The standard evaluation measure is the PARSEVAL metric
- It measures how much of the *constituents* are correct, meaning:
  - the constituent spans the same words as in the gold standard
  - it has the same label (*NP*, *VP*, etc.)
- Correctness is given by the  $F_1$ -score. The state of the art is 92-93%.

# Problems with (P)CFG: #1

Problem: it does not support subcategorization.

- (1) John ate a cake.
- (2) \*John slept a cake.

These two sentences were generated by the same  $VP \rightarrow \text{Verb NP}$  rule.

Solution: **Lexicalization**

- Annotate constituents with its lexical head:
  - $VP[\text{eat}] \rightarrow \text{Verb}[\text{eat}] \text{ NP}$  will appear during training
  - $VP[\text{sleep}] \rightarrow \text{Verb}[\text{sleep}] \text{ NP}$  will not
- Introduces the **sparsity problem**
  - Smoothing: interpolate with the raw probability
$$P(VP[\text{eat}] \rightarrow \text{Verb}[\text{eat}] \text{ NP}) + \lambda P(VP \rightarrow \text{Verb NP})$$
  - Better handled via semantic methods

# Problems with (P)CFG: #2

Problem:

- (P)CFG has strong independence assumptions: a rule can be applied whenever its left-hand nonterminal is present, irrespective of context
- Language is full of interdependence:

- (1) The dog was chasing the cat.
- (2) \*The dog were chasing the cat.

Solution: annotate constituents with grammatical features, and propagate them up the tree

- E.g. number and person
- A sentence is grammatical iff the features agree (*dog* and *was* are both [Sg])
- This technique is **mildly context sensitive**

# Appendix: bibliography



Chomsky, Noam (1956). “Three models for the description of language”. In: *IRE Transactions on Information Theory* 2, pp. 113–124.