# 1st Tutorial: Introduction and Grammar Induction

## Practical Course in Natural Language Processing

Boris Petrov

2026-04-21

# Overview


1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

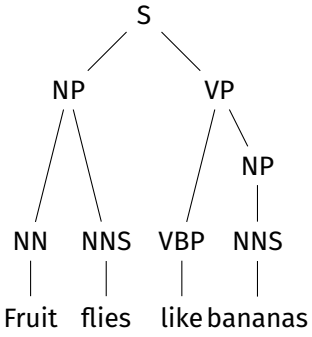
# Outline


1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

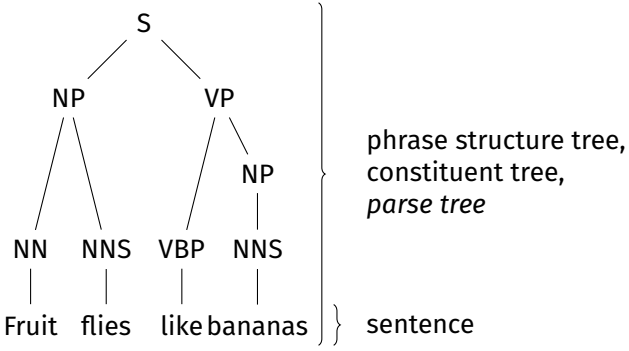
# Parsing (natural language processing / NLP)

Fruit flies like bananas

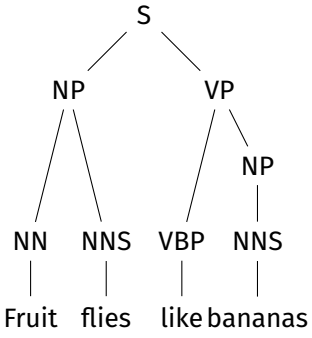
# Parsing (natural language processing / NLP)



# Parsing (natural language processing / NLP)

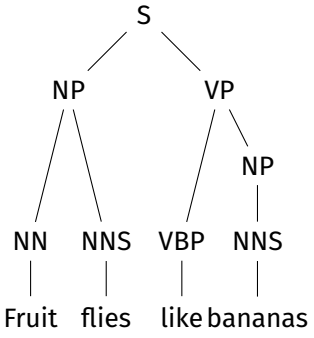


# Parsing (natural language processing / NLP)




Goal: automate this annotation

# Parsing (natural language processing / NLP)

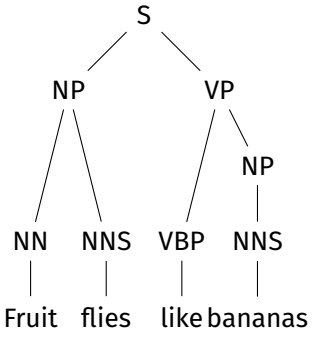



Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

# Parsing (natural language processing / NLP)



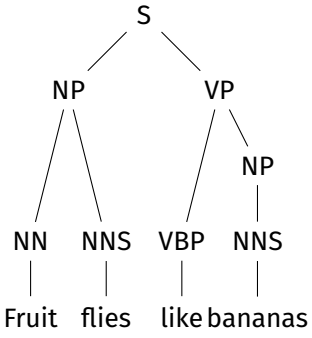

Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let S = S and give G by its rules.*

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

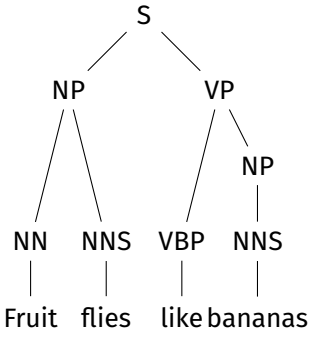
- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let S = S and give G by its rules.*

Read off the rules of the constituent tree:

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

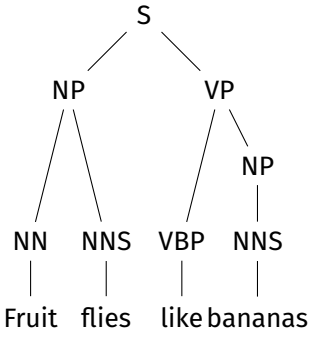
$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let S = S and give G by its rules.*

Read off the rules of the constituent tree:

$$S \to NP\ VP$$

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

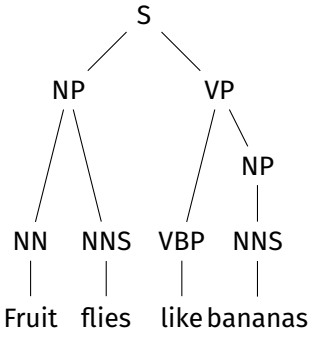
$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let* S = S *and give G by its rules.*

Read off the rules of the constituent tree:

$$\mathrm{S} \to \mathrm{NP\ VP} \qquad \mathrm{NP} \to \mathrm{NN\ NNS}$$

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

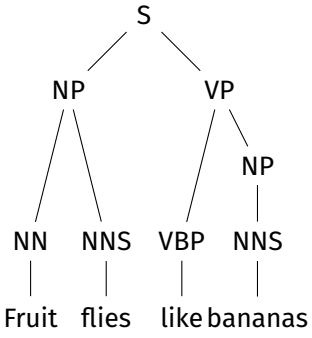
$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let* S = S *and give G by its rules.*

Read off the rules of the constituent tree:

$\mathrm{S} \to \mathrm{NP\ VP}$ $\qquad$ $\mathrm{NP} \to \mathrm{NN\ NNS}$ $\qquad$ $\mathrm{VP} \to \mathrm{VBP\ NP}$

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

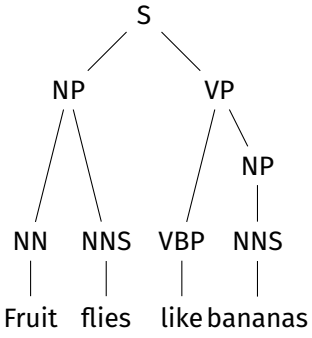
$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let* S = S *and give G by its rules.*

Read off the rules of the constituent tree:

$S \to NP\ VP$ $\quad$ $NP \to NN\ NNS$ $\quad$ $VP \to VBP\ NP$ $\quad$ $NP \to NNS$

# Parsing (natural language processing / NLP)




Context-free grammar (CFG):
$G = (N, \Sigma, S, R)$

- $N$ finite set (*nonterminals*)
- $\Sigma$ finite set (*terminals*), $N \cap \Sigma = \emptyset$,
- $S \in N$ (*initial nonterminal*)
- $R$ finite set (*rules*) of the form

$$A \to \alpha \qquad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

*We let S = S and give G by its rules.*

Read off the rules of the constituent tree:

$S \to NP\ VP$ $\quad$ $NP \to NN\ NNS$ $\quad$ $VP \to VBP\ NP$ $\quad$ $NP \to NNS$

$NN \to fruit$ $\quad$ $NNS \to flies$ $\quad$ $VBP \to like$ $\quad$ $NNS \to bananas$

# Parsing (natural language processing / NLP)

Parse a new sentence with our grammar

Bananas like fruit flies

Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow fruit$ $NNS \rightarrow flies$ $VBP \rightarrow like$ $NNS \rightarrow bananas$

# Parsing (natural language processing / NLP)

CYK algorithm:
derive the initial nonterminal
bottom-up

Bananas like fruit flies

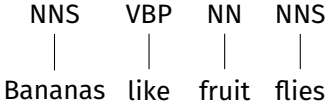
Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow fruit$ $NNS \rightarrow flies$ $VBP \rightarrow like$ $NNS \rightarrow bananas$

# Parsing (natural language processing / NLP)

CYK algorithm:
derive the initial nonterminal
bottom-up

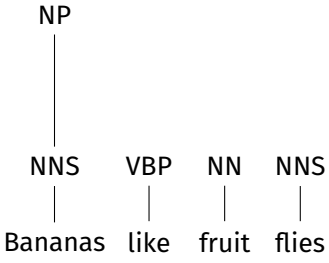



Read off the rules of the constituent tree:

$S \to NP\ VP$ $NP \to NN\ NNS$ $VP \to VBP\ NP$ $NP \to NNS$

$NN \to fruit$ $NNS \to flies$ $VBP \to like$ $NNS \to bananas$

# Parsing (natural language processing / NLP)



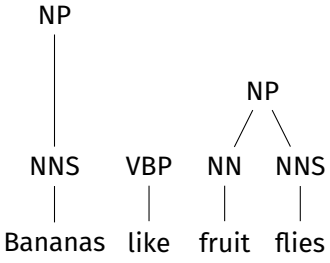

CYK algorithm:
derive the initial nonterminal bottom-up

Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow fruit$ $NNS \rightarrow flies$ $VBP \rightarrow like$ $NNS \rightarrow bananas$

# Parsing (natural language processing / NLP)



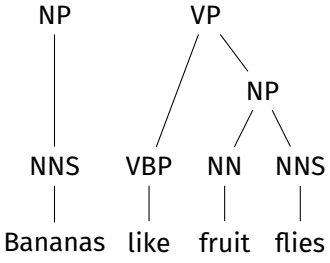

CYK algorithm:
derive the initial nonterminal bottom-up

Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow fruit$ $NNS \rightarrow flies$ $VBP \rightarrow like$ $NNS \rightarrow bananas$

# Parsing (natural language processing / NLP)



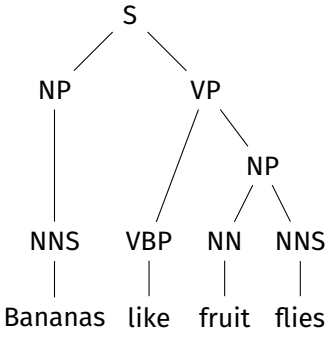

CYK algorithm:
derive the initial nonterminal bottom-up

Read off the rules of the constituent tree:

$S \to NP\ VP$ $NP \to NN\ NNS$ $VP \to VBP\ NP$ $NP \to NNS$

$NN \to fruit$ $NNS \to flies$ $VBP \to like$ $NNS \to bananas$

# Parsing (natural language processing / NLP)




CYK algorithm:
derive the initial nonterminal bottom-up

Read off the rules of the constituent tree:

$S \to NP\ VP$ $NP \to NN\ NNS$ $VP \to VBP\ NP$ $NP \to NNS$

$NN \to fruit$ $NNS \to flies$ $VBP \to like$ $NNS \to bananas$

# Parsing (natural language processing / NLP)

Problem: several rules may be applicable

Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow$ fruit $NNS \rightarrow$ flies $VBP \rightarrow$ like $NNS \rightarrow$ bananas

# Parsing (natural language processing / NLP)

Problem: several rules may be applicable

Solution: Probabilistic CFG (PCFG)

Read off the rules of the constituent tree:

$S \rightarrow NP\ VP$ $NP \rightarrow NN\ NNS$ $VP \rightarrow VBP\ NP$ $NP \rightarrow NNS$

$NN \rightarrow fruit$ $NNS \rightarrow flies$ $VBP \rightarrow like$ $NNS \rightarrow bananas$

# Parsing (natural language processing / NLP)

PCFG: $(G, p)$ where $p\colon R \to [0,1]$, *proper* if

$$\forall A \in N\colon \sum_{r=(A\to\alpha)\in P} p(r) = 1$$

probability of parse tree $d = r_1 \cdots r_n$

$$P(d \mid (G,p)) = p(r_1) \cdot \cdots \cdot p(r_n)$$

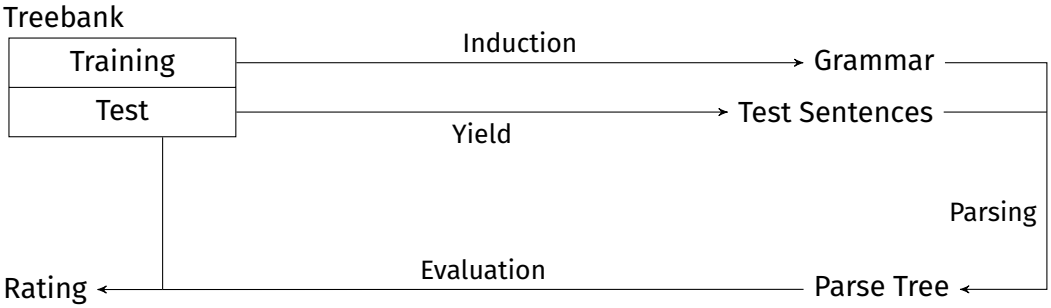
probability of sentence $w \in \Sigma^*$

$$P(w \mid (G,p)) = \sum_{d \text{ parse tree of } w} P(d \mid (G,p))$$

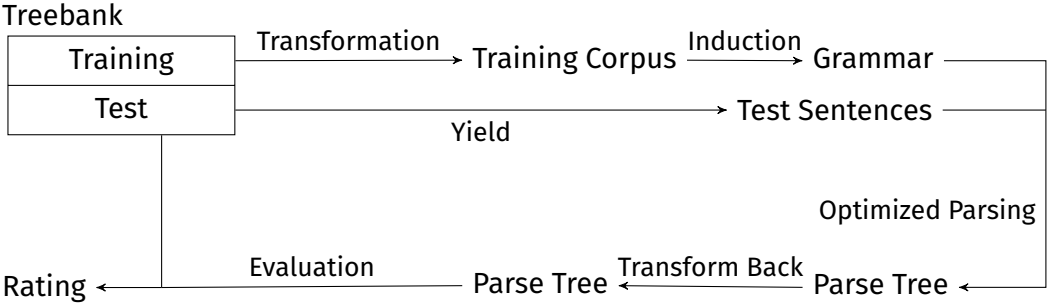
# Outline


1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

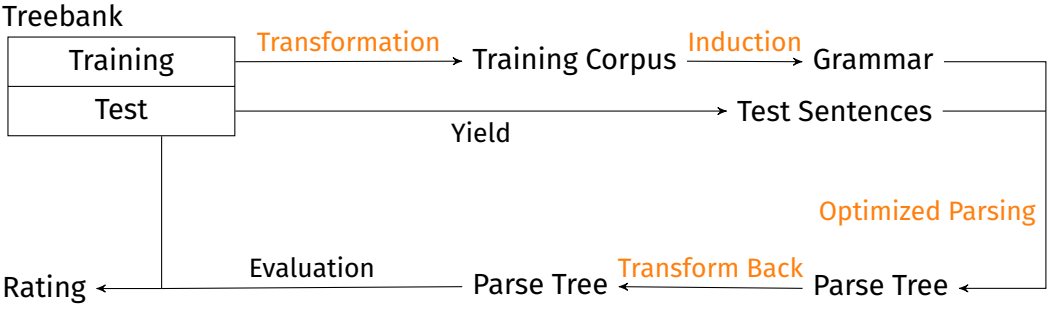
# Objective


Treebank
Training
Test
Induction
Grammar
Test Sentences
Yield
Parsing
Evaluation
Rating
Parse Tree

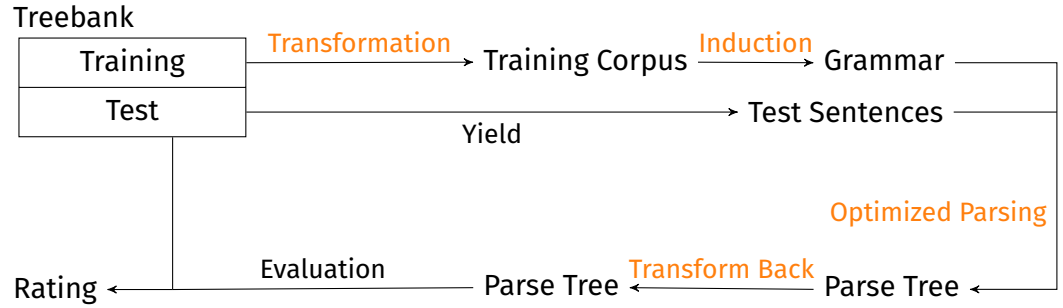
# Objective


Treebank
Training
Test
Transformation
Training Corpus
Induction
Grammar
Test Sentences
Yield
Optimized Parsing
Rating
Evaluation
Parse Tree
Transform Back
Parse Tree

# Objective


Treebank
Training
Test
Transformation
Training Corpus
Induction
Grammar
Test Sentences
Yield
Optimized Parsing
Rating
Evaluation
Parse Tree
Transform Back
Parse Tree

# Objective




Criterias

- Rating/F1-Measure
- Speed

# Problem Statement

Details: see the schedule and descriptions of the individual tasks
Required tasks:

1. Induction of the grammar
2. Parser (best parse tree) – Bottom-up (CYK) *or* deductive (Knuth's algorithm)
3. Corpus transformations: trivial unking and debinarization

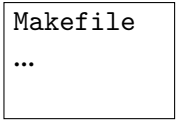
Optional tasks (min. 3):

3. Corpus transformations
   - Binarization and Markovization
   - Smoothing of the induced grammar
4. Parser optimization
   - Pruning
   - Heuristic search (A*)

Seminar/oral exam (depending on the module)

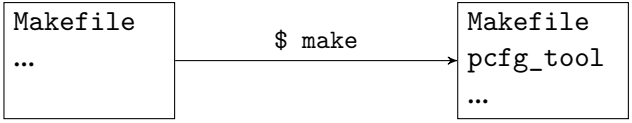
# Assessment

Code-Repository



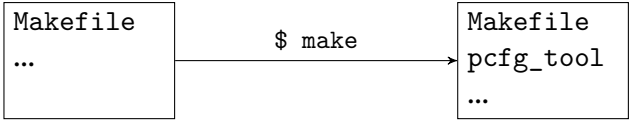
# Assessment

Code-Repository



# Assessment

Code-Repository




- `pcfg_tool` is executable
- `pcfg_tool` provides solutions for *all* tasks via sub-commands (like git)
- Sub-task not solved? → Exit-Code 22 in the sub-command/argument
- Full command-line interface in the schedule
- Verification of functionality through integration tests
- Automated ranking in a competitive environment

# Outline


1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

# Dataset

Penn Treebank Wall Street Journal [1]

Training corpus: Sections 00–18

- `training.mrg` – Constituent trees for grammar induction

Already induced grammars

- `grammar.rules`
- `grammar.lexicon`
- `grammar.words`

} induced from binarized training corpus

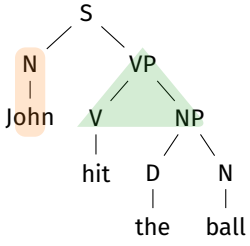
Testkorpus: Sections 19–21

- `gold.mrg` – Constituent trees for evaluation
- `gold_b.mrg` – Binarized constituent trees for evaluation
- `testsentences` – Yield of the test corpus for evaluation

# Format of the constituent trees

(about `training.mrg`, `gold.mrg`, `gold_b.mrg`)

- 1 line = 1 constituent tree ⇒ stream processing (also in general)
- Trees are stored as *symbolic expressions* (s-expressions)




→ (S (N John)(VP (V hit) (NP (D the) (N ball))))

Inductive definition:

Atom: Strings that contain neither spaces nor parentheses

List: $(s_1 \_ s_2 \_ \cdots \_ s_k)$ s.t. $s_i$ are also s-expressions ($\_ \hat{=}$ space)

# Implications of the format

Trees as s-expressions:

- No empty lists (at least one leaf)
- $s_1$ is always an atom (node label)

Implications for symbols (non-terminals)

- No spaces
- Escaping parentheses (-LRB-, -RRB-)

Sentences: are *tokenized*, i.e.
"There's no food", he says. → " There 's no food " , he says .

Watch out for stray spaces: $(\_A\_(\_s_1\_s_2\_ \ldots \_s_k\_)\_)$

# Outline

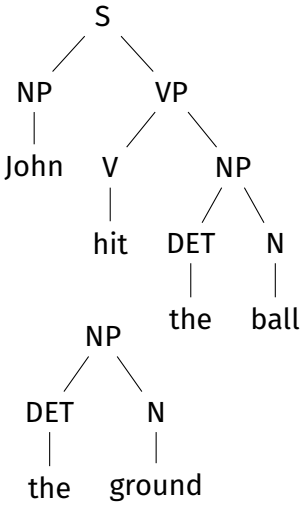

1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

# Algorithm (pseudo-code)

1: **function** INDUCE_GRAMMAR(corpus $c = t_1, \ldots, t_n$)
2: **for** $i = 1, \ldots, n$ **do**
3: **for each** position $p$ of $t_i$ **do**
4: $r \leftarrow$ readoff rule at $p$
5: increase count of $r$ by 1
6: $R \leftarrow$ set of all counted rules
7: $N, \Sigma \leftarrow$ all nonterminals/terminals occurring in $R$
8: $S$ depends on corpus (command line argument)
9: $p \leftarrow$ normalized counts of rules with same left-hand side
10: **return** $((N, \Sigma, S, R), p)$

# Algorithm (illustration)

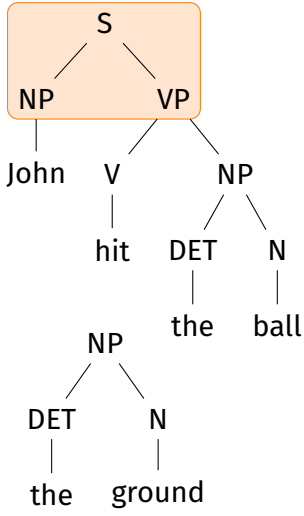
Rules:



# Algorithm (illustration)

Rules:

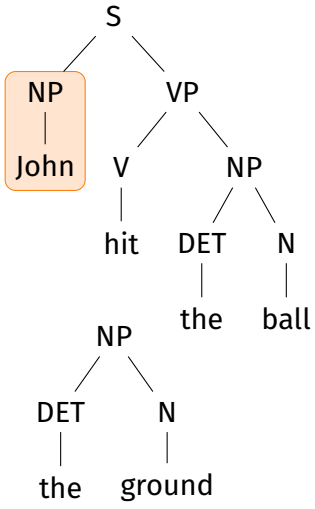
- S → NP VP # 1



# Algorithm (illustration)

Rules:

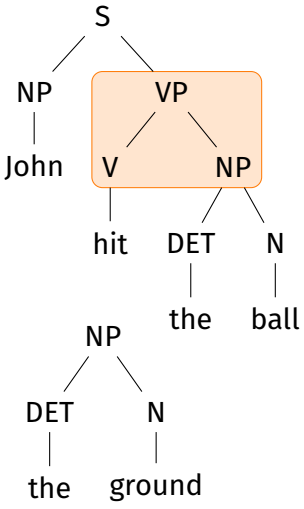
- S → NP VP # 1
- NP → John # 1



# Algorithm (illustration)

Rules:

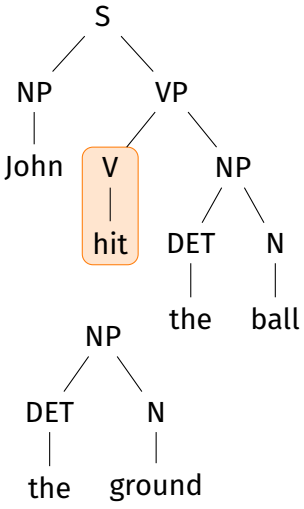
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1



# Algorithm (illustration)

Rules:

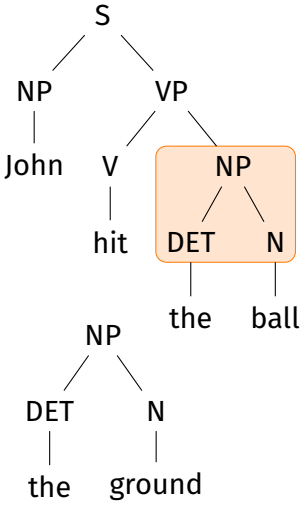
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1



# Algorithm (illustration)

Rules:

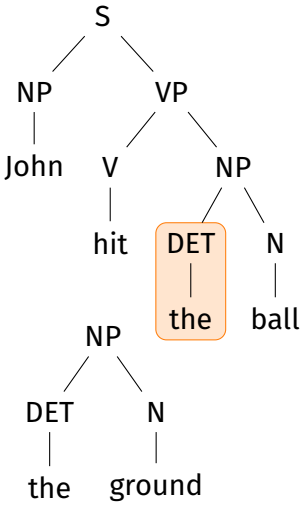
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 1



# Algorithm (illustration)

Rules:

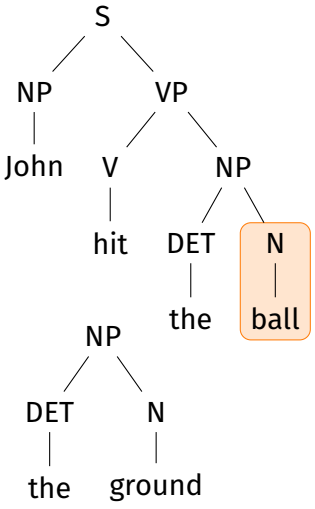
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 1
- DET → the # 1



# Algorithm (illustration)

Rules:

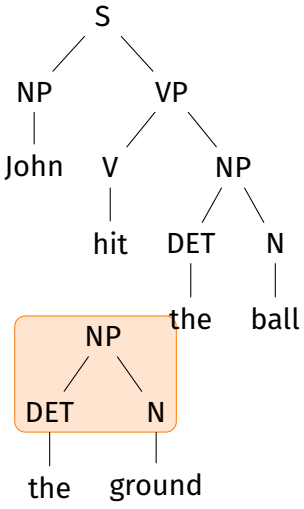
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 1
- DET → the # 1
- N → ball # 1



# Algorithm (illustration)

Rules:

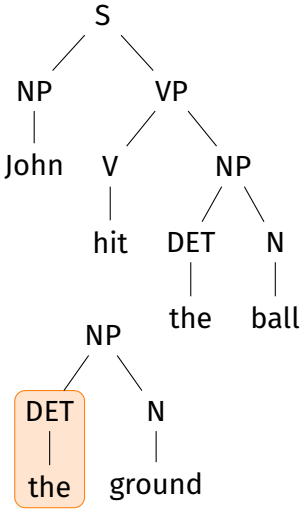
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 1
- N → ball # 1



# Algorithm (illustration)

Rules:

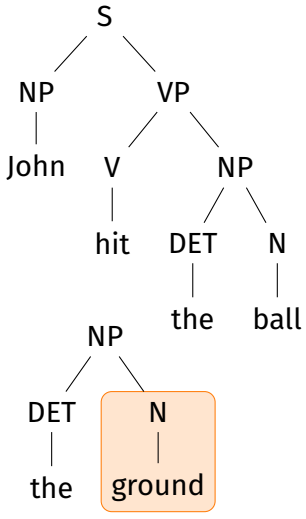
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1



# Algorithm (illustration)

Rules:

- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1
- N → ground # 1



# Algorithm (illustration)

Rules:

- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1
- N → ground # 1

Normalized rules:

- S → NP VP # 1
- VP → V NP # 1
- V → hit # 1

# Algorithm (illustration)

Rules:

- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1
- N → ground # 1

Normalized rules:

- S → NP VP # 1
- NP → John # $\frac{1}{1+2} = 1/3$
- VP → V NP # 1
- V → hit # 1
- NP → DET N # $\frac{2}{1+2} = 2/3$

# Algorithm (illustration)

Rules:

- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1
- N → ground # 1

Normalized rules:

- S → NP VP # 1
- NP → John # $\frac{1}{1+2} = 1/3$
- VP → V NP # 1
- V → hit # 1
- NP → DET N # $\frac{2}{1+2} = 2/3$
- DET → the # $\frac{2}{2} = 1$

# Algorithm (illustration)

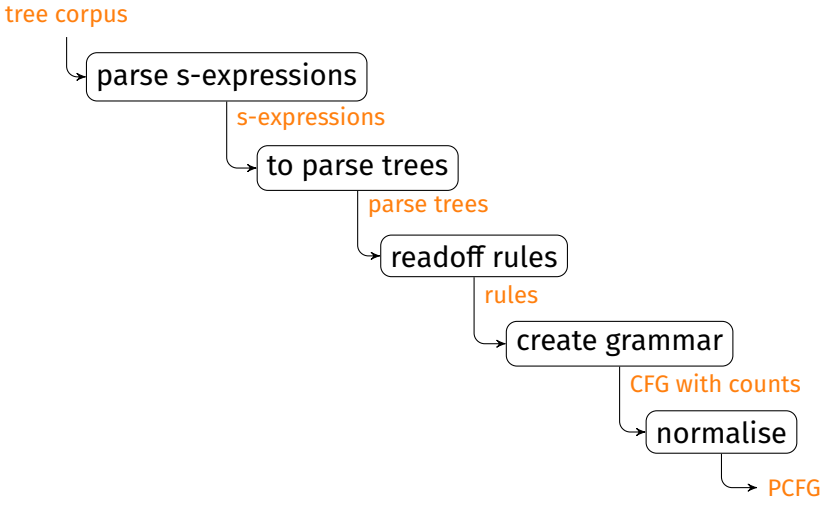
Rules:

- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1
- N → ground # 1

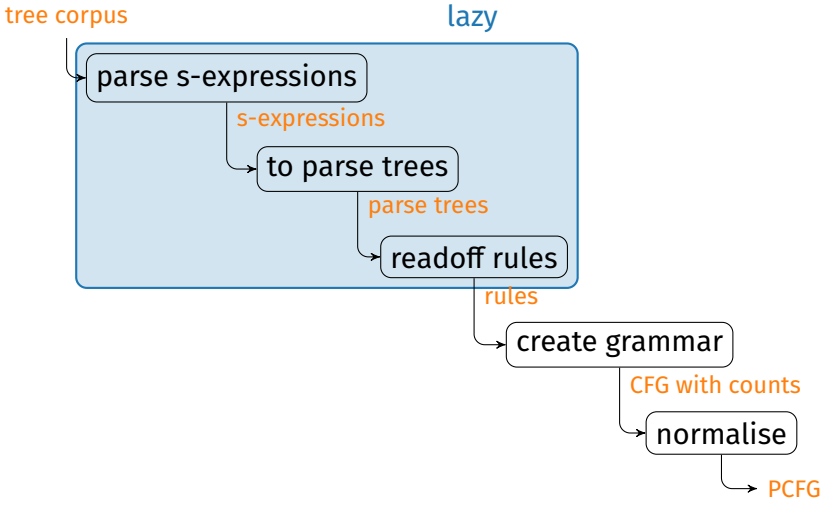
Normalized rules:

- S → NP VP # 1
- NP → John # $\frac{1}{1+2} = 1/3$
- VP → V NP # 1
- V → hit # 1
- NP → DET N # $\frac{2}{1+2} = 2/3$
- DET → the # $\frac{2}{2} = 1$
- N → ball # $\frac{1}{1+1} = 1/2$
- N → ground # $\frac{1}{1+1} = 1/2$

# Suggestion: implementation as a pipeline



# Suggestion: implementation as a pipeline



# Data structures (suggestion)

- s-expression:

```
pub enum SExp<A> {
    Atom(A),
    List(Vec<SExp<A>>),
}
```

`A` could simply be a `String`

# Data structures (suggestion)

- s-expression:

```
pub enum SExp<A> {
    Atom(A),
    List(Vec<SExp<A>>),
}
```

- Baum

```
pub struct Tree<A> {
  pub root: A,
  pub children: Vec<Tree<A>>
}
```

`A` could simply be a `String`

# Data structures (suggestion)

- Regel

```rust
pub enum Rule<N, T> {
  Lexical    { lhs: N, rhs: T,      },
  NonLexical { lhs: N, rhs: Vec<N>, },
}
```

`N` and `T` could also be `String`s, `W` e.g. `f64`

# Data structures (suggestion)

- Regel

```
pub enum Rule<N, T> {
  Lexical    { lhs: N, rhs: T,      },
  NonLexical { lhs: N, rhs: Vec<N>, },
}
```

- Grammatik

```
pub struct Grammar<N, T, W> where N: Eq + Hash, T: Eq + Hash {
  initial: N,
  rules: HashMap<Rule<N, T>, W>,
}
```

`N` and `T` could also be `String`s, `W` e.g. `f64`

# General advice

- Use *Unit-Tests*

# General advice

- Use *Unit-Tests*
- Use a *Build-System*, e.g. cargo (Rust), stack or cabal (Haskell)

# General advice

- Use *Unit-Tests*
- Use a *Build-System*, e.g. cargo (Rust), stack or cabal (Haskell)
- Use a *CLI-Library*, e.g. clap (Rust), cmdargs (Haskell)

# General advice

- Use *Unit-Tests*
- Use a *Build-System*, e.g. cargo (Rust), stack or cabal (Haskell)
- Use a *CLI-Library*, e.g. clap (Rust), cmdargs (Haskell)
- Use a *Parser-Combinator-Library*, e.g. nom (Rust), parsec (Haskell), parsy (Python)

# General advice

- Use *Unit-Tests*
- Use a *Build-System*, e.g. cargo (Rust), stack or cabal (Haskell)
- Use a *CLI-Library*, e.g. clap (Rust), cmdargs (Haskell)
- Use a *Parser-Combinator-Library*, e.g. nom (Rust), parsec (Haskell), parsy (Python)
- Use *Iterators* to work with big data structures

# Outline


1. Introduction to Parsing
2. Practical Course Content
3. Material
4. Grammar Induction
5. Organisation

# Schedule

Timeslot: Tuesday, 2nd block (9:20-10:50), APB/E042

Tutorials (attendance required) on the following dates:

- 21. 04. Course presentation, introduction to task 1 (grammar induction)
- 12. 05. Evaluation of task 1, introduction to task 2 (parsing)
- 16. 06. Evaluation of task 2, introduction to task 3 (corpus transformation)
- 30. 06. Introduction to task 4 (parser optimization)
- 28. 07. Evaluation of tasks 3 and 4, competition

For other dates: *opportunity* for independent study

Submit your assignments by **Friday, 12:00 p.m. CEST** during the week before grading – strongly recommended to submit them earlier!

# Out-of-class organization

Testing of the interfaces at `ganymed.inf.tu-dresden.de` (ZIH-Login via SSH)

Asynchronous communication by

- Questions
- Submission of tasks
- Feedback

`boris.petrov@tu-dresden.de`

# References I

[1] Mitchell Marcus et al. “The Penn Treebank: Annotating Predicate Argument Structure”. HLT ’94. 1994.