Tutorium 1: Grammar Induction

Praktikum Verarbeitung natürlicher Sprache

Richard Mörbitz

2021-04-19

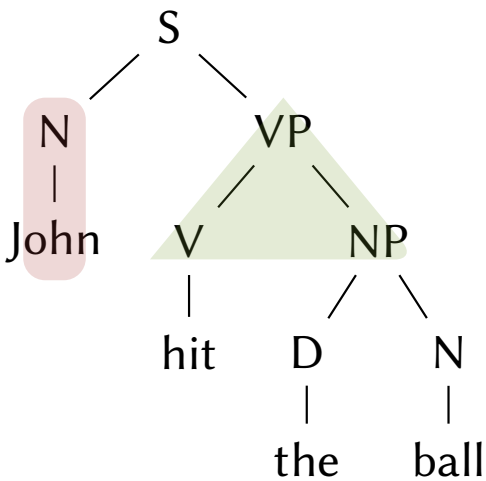
Overview

1. Introduction to the training corpus
2. Algorithmic approach
3. Overall pipeline (suggestion)
4. Data structures (suggestion)
5. General advice

Introduction to the training corpus

Take a look at `training.mrg`:

- one line = one constituent tree (annotated sentence)
- trees are stored as *symbolic expressions* (s-expressions)



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

Inductive definition of s-expressions:

atoms: strings that do not contain spaces or parenthesis

lists: $(s_1 \sqcup s_2 \sqcup \ldots \sqcup s_k)$ where s_i are s-expressions ($\sqcup$ is space)

Introduction to the training corpus: some remarks

Trees as s-expressions:

- no empty lists
- s_1 is always an atom

Parenthesis in grammar symbols are escaped (-LRB-, -RRB-).

Sentences are *tokenized*, e.g.,

> *“There’s no food”, he says.*

becomes

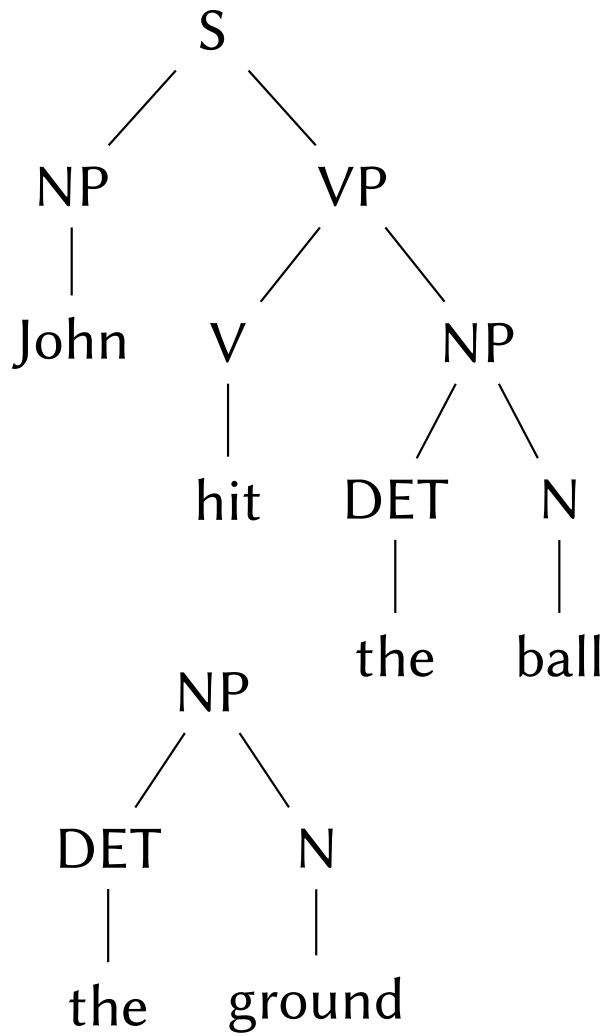
> *“ There ’s no food ” , he says .*

Algorithmic approach

1: **function** INDUCE_GRAMMAR(corpus $c = t_1, \dots, t_n$)
2: **for** $i = 1, \dots, n$ **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 (usually ROOT)
9: $p \leftarrow$ normalized counts of rules with same left-hand side
10: **return** $((N, \Sigma, S, R), p)$

Algorithmic approach

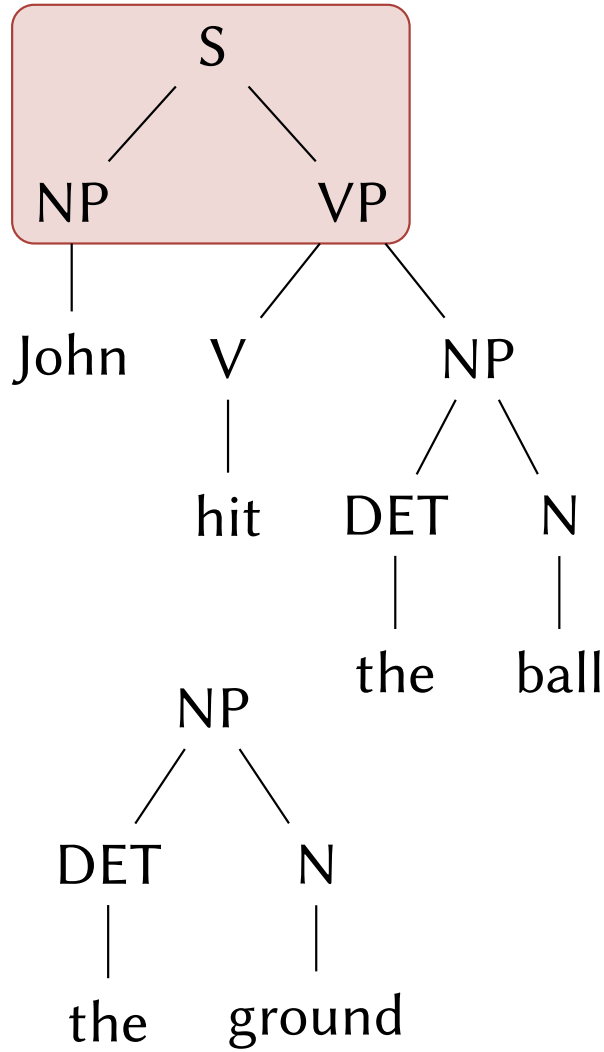
rules:



Algorithmic approach

rules:

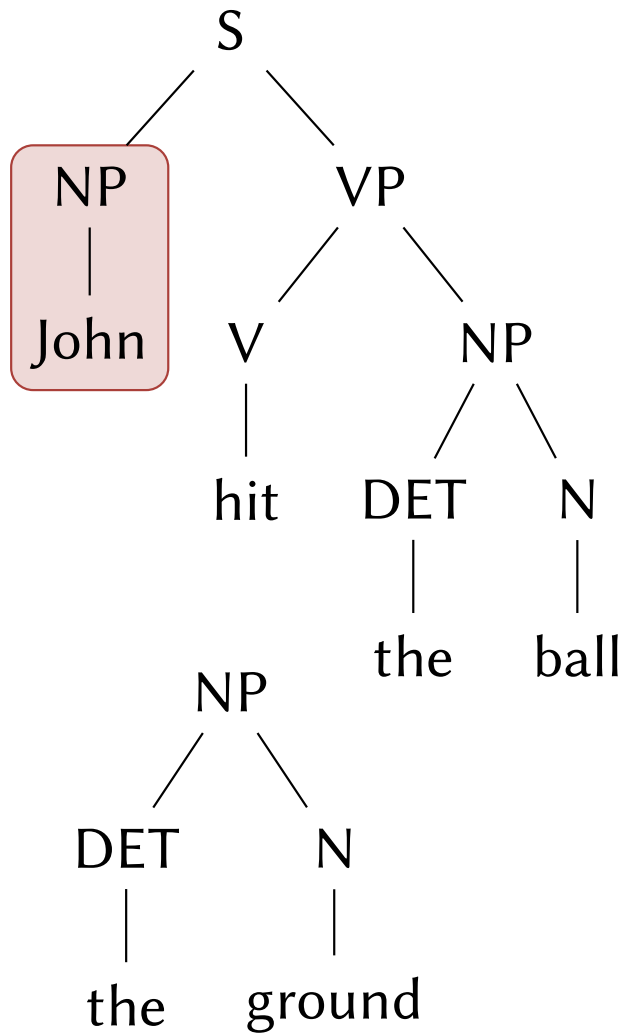
- S → NP VP



Algorithmic approach

rules:

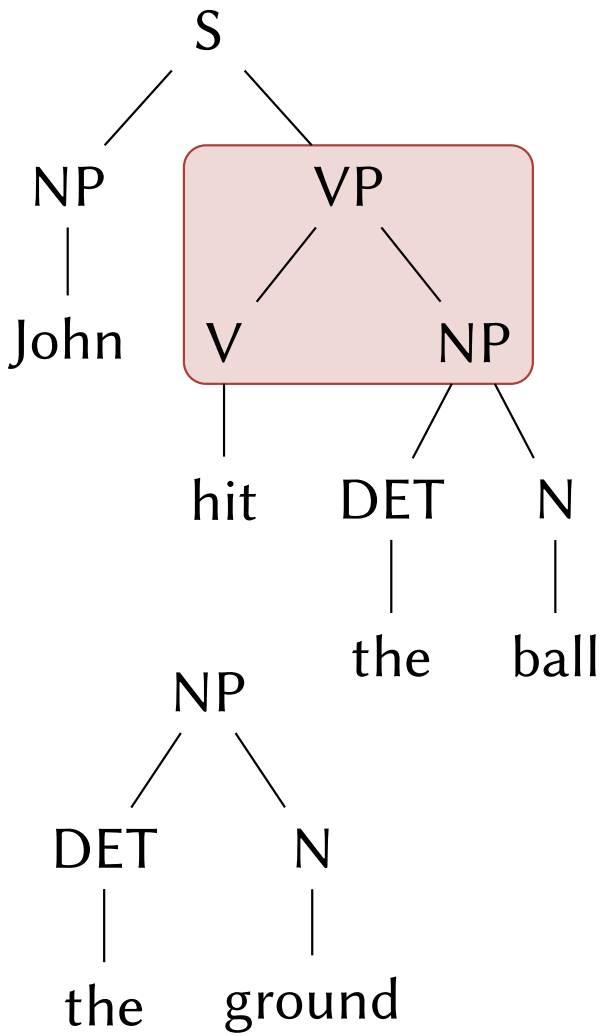
- S → NP VP
- NP → John



Algorithmic approach

rules:

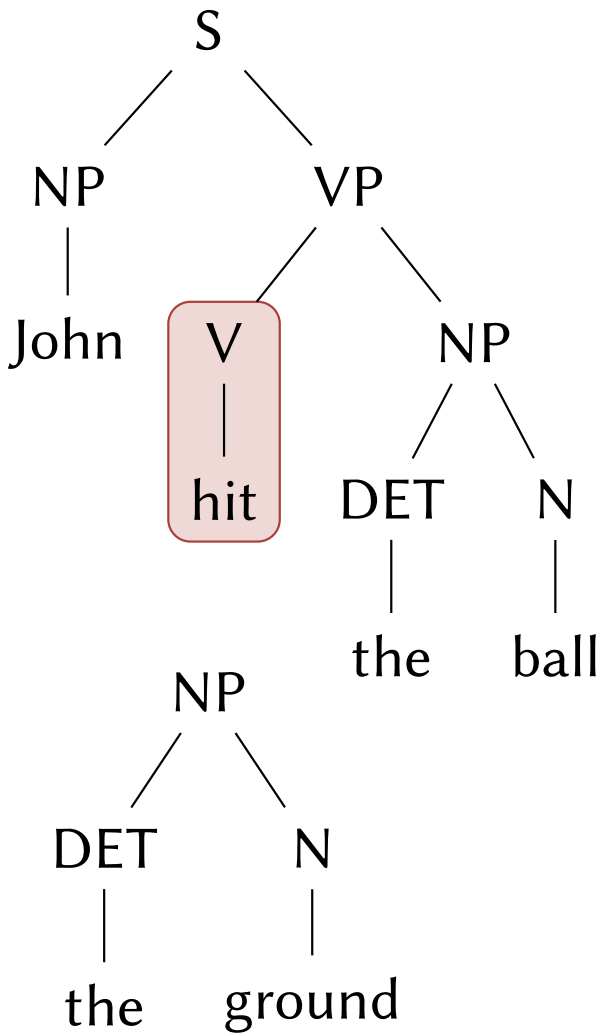
- S → NP VP
- NP → John
- VP → V NP



Algorithmic approach

rules:

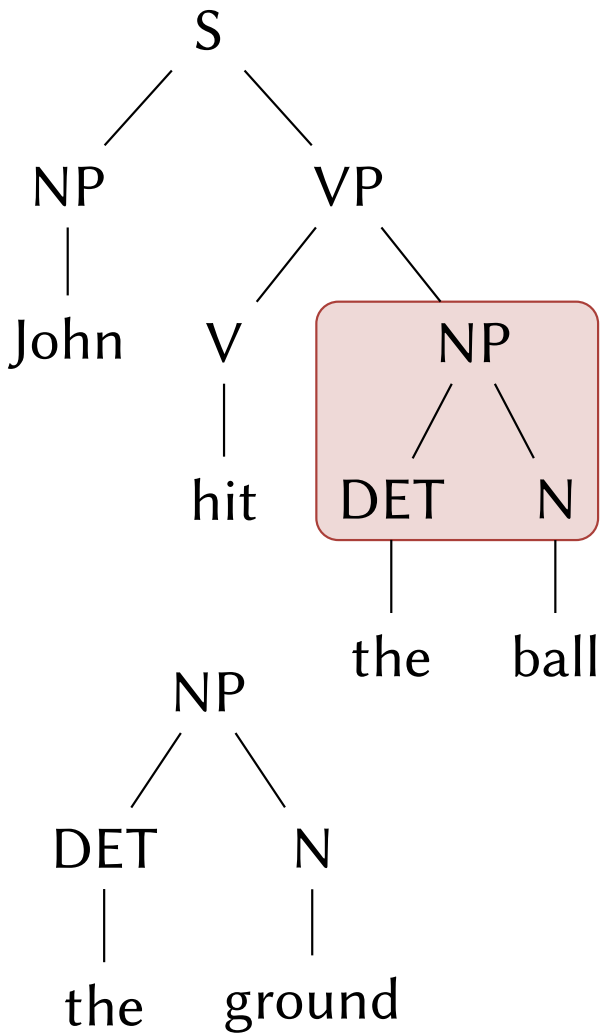
- S → NP VP
- NP → John
- VP → V NP
- V → hit



Algorithmic approach

rules:

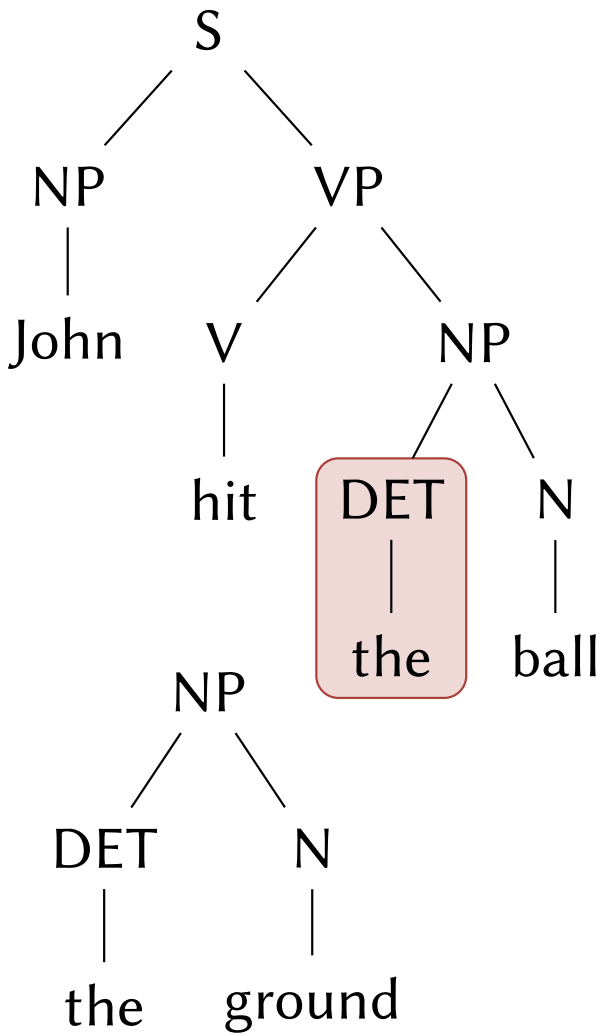
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N



Algorithmic approach

rules:

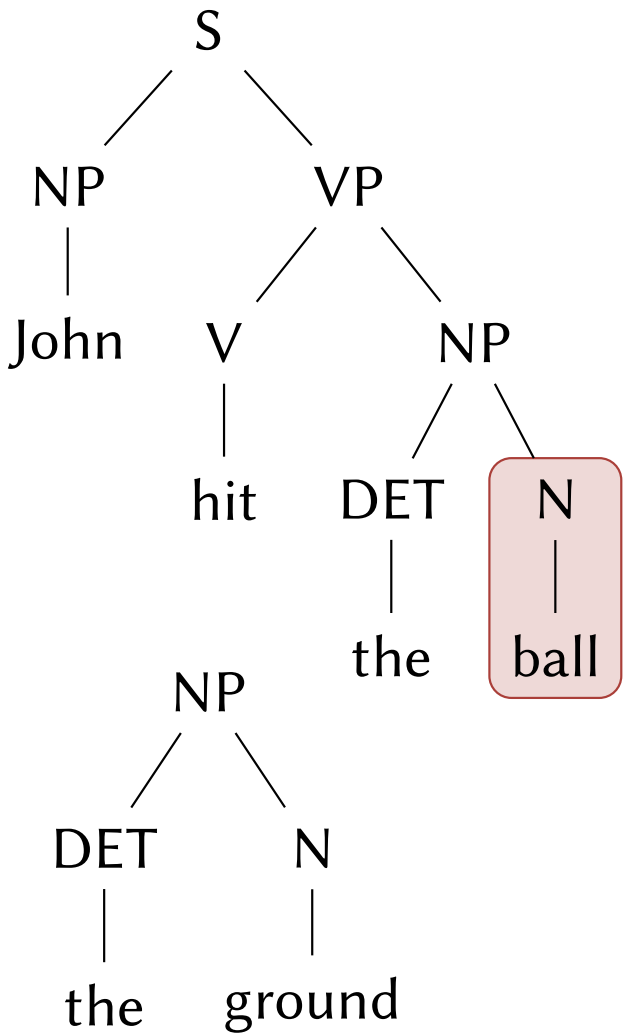
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N
- DET → the



Algorithmic approach

rules:

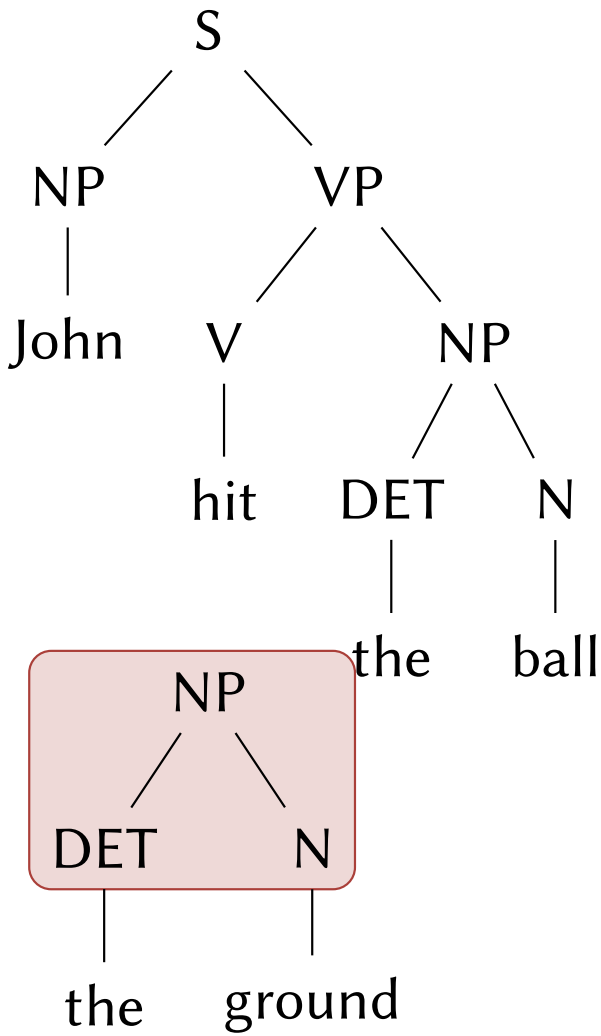
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N
- DET → the
- N → ball



Algorithmic approach

rules:

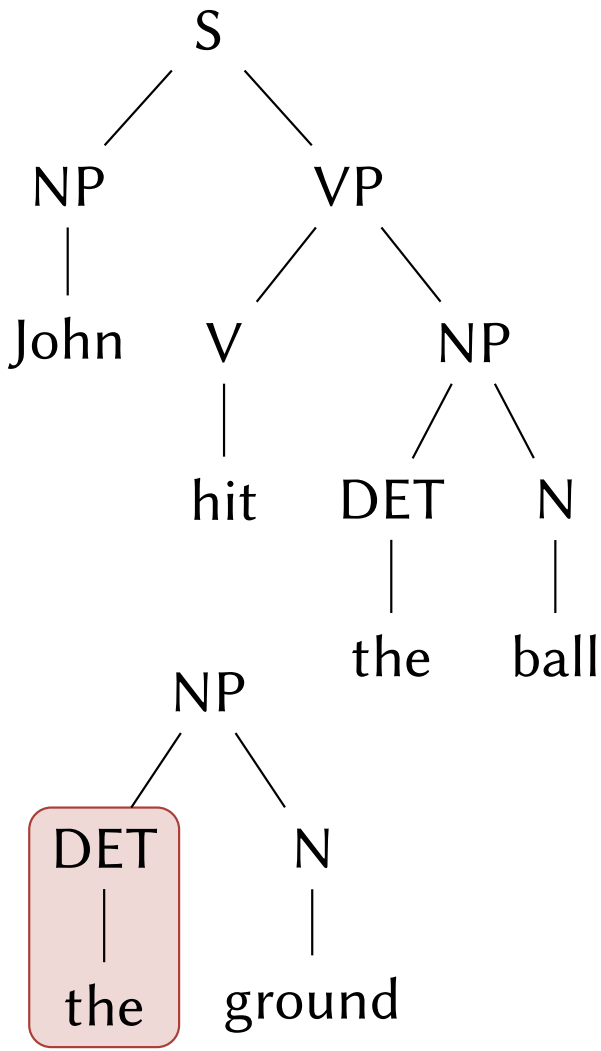
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 1
- N → ball # 1



Algorithmic approach

rules:

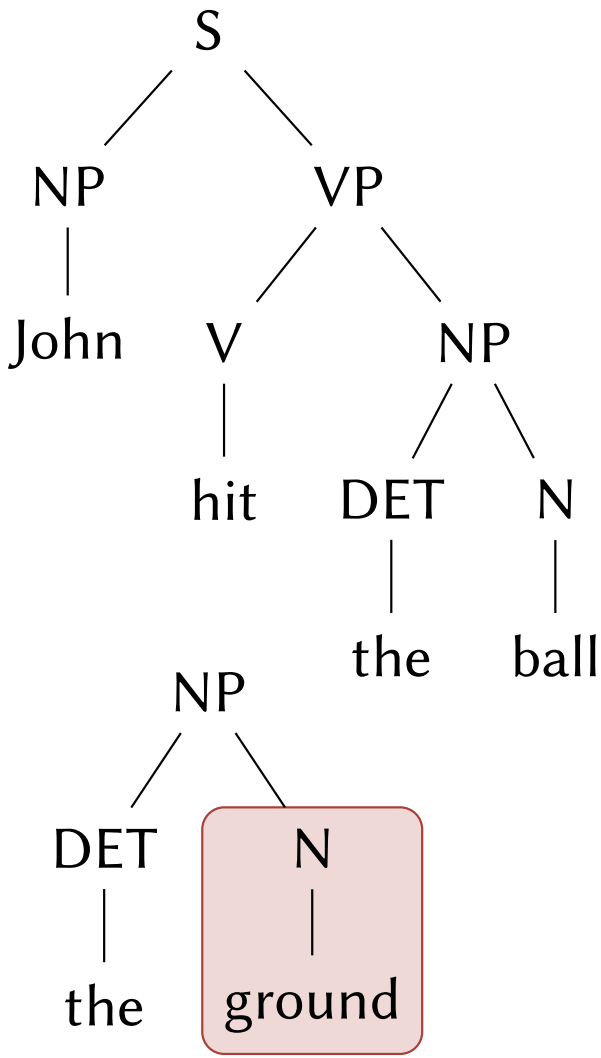
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1



Algorithmic approach

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



Algorithmic approach

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

normalised rules:

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

Algorithmic approach

rules:

- $S \rightarrow NP\ VP$ $\quad \# 1$
- $NP \rightarrow John$ $\quad \# 1$
- $VP \rightarrow V\ NP$ $\quad \# 1$
- $V \rightarrow hit$ $\quad \# 1$
- $NP \rightarrow DET\ N$ $\quad \# 2$
- $DET \rightarrow the$ $\quad \# 2$
- $N \rightarrow ball$ $\quad \# 1$
- $N \rightarrow ground$ $\quad \# 1$

normalised rules:

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

Algorithmic approach

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

normalised 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 # $\frac{2}{2} = 1$

Algorithmic approach

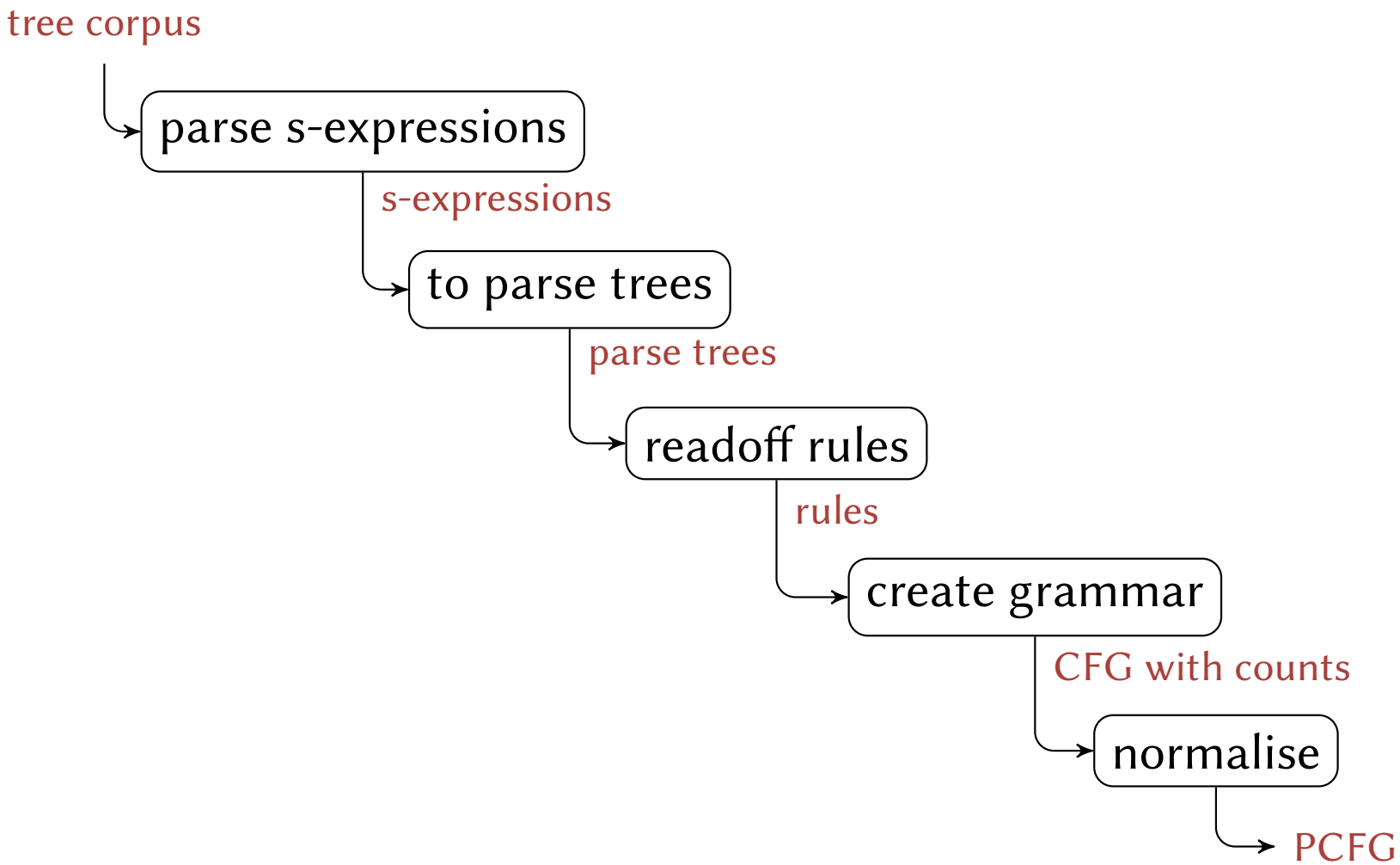
rules:

- 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

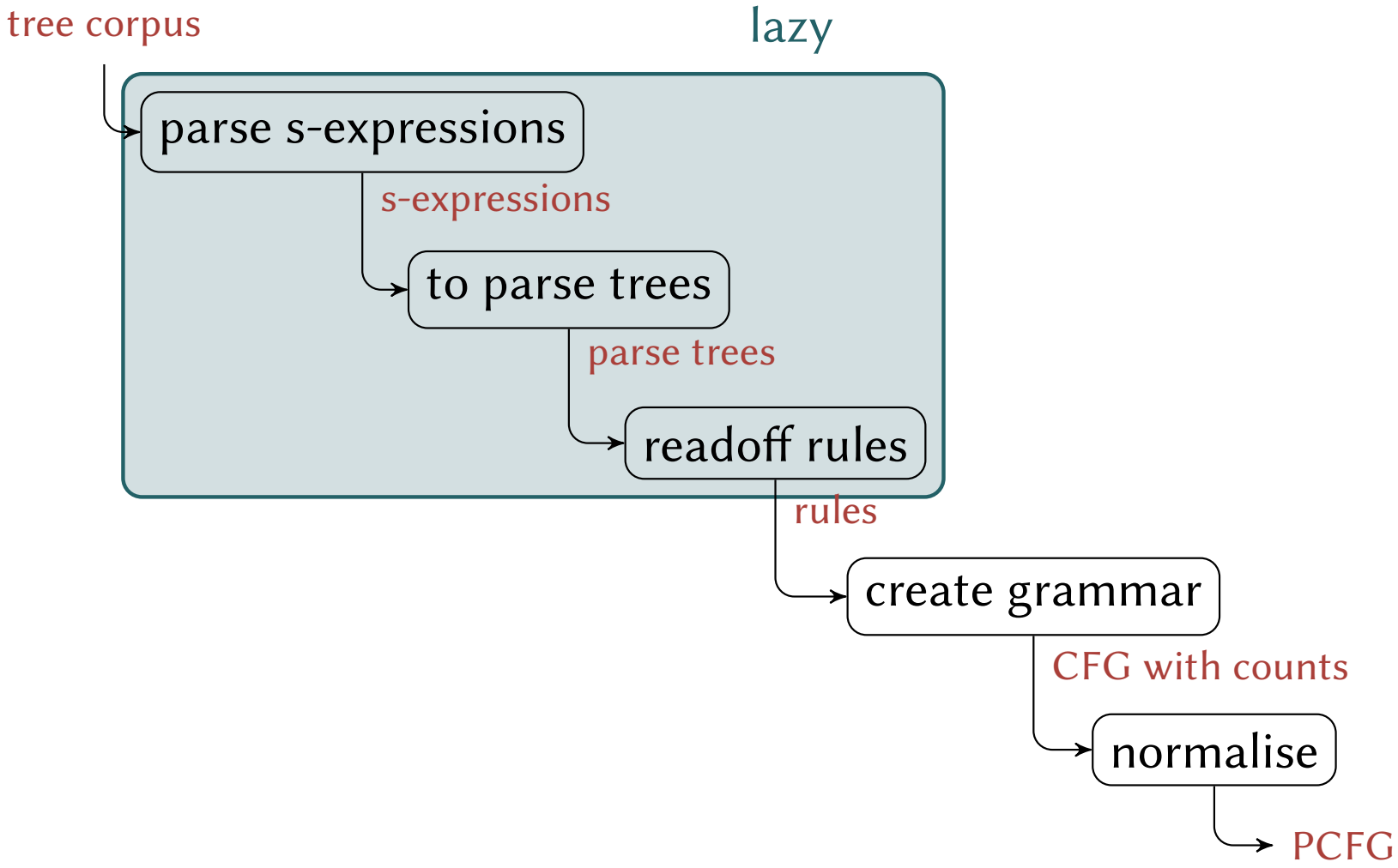
normalised rules:

- 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 # $\frac{1}{1+1} = 1/2$

Overall pipeline (suggestion)



Overall pipeline (suggestion)



Data structures (suggestion)

- s-expression:

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

A can be chosen the string type

Data structures (suggestion)

- s-expression:

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

- tree

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

A can be chosen the string type

Data structures (suggestion)

- rule

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

`N` and `T` can be chosen as the string type, `W` as a floating point type

Data structures (suggestion)

- rule

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

- grammar

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

`N` and `T` can be chosen as the string type, `W` as a floating point type

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)

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)
- use *iterators* to act lazily on large data (e.g. corpora)