Induction of probabilistic context-free grammars from tree corpora

Praktikum Parsing von natürlichen Sprachen

Tobias Denkinger

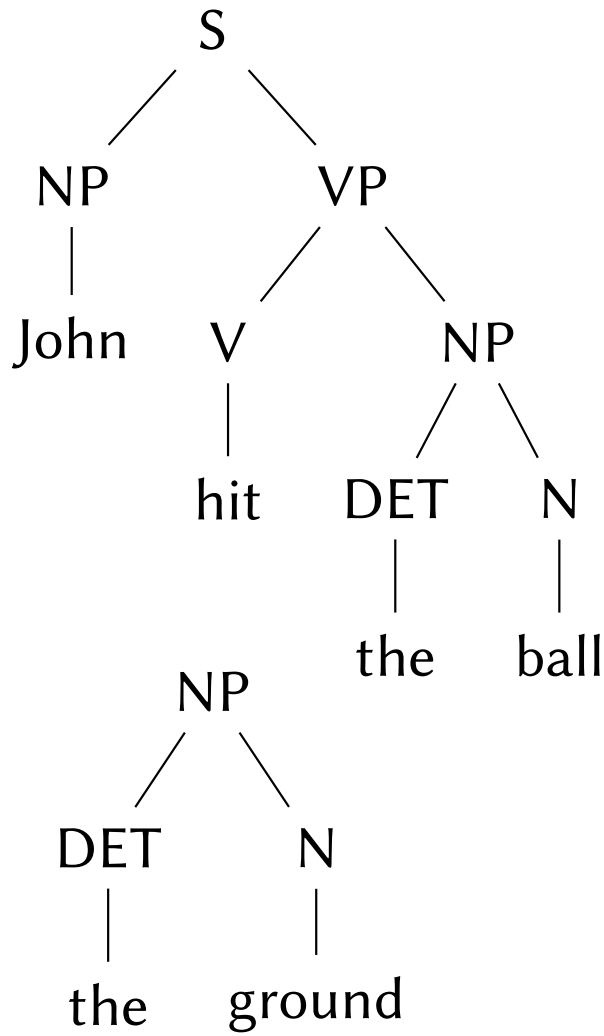
2019-04-09

Overview

1. Idea (recap)
2. Overall pipeline (suggestion)
3. Trees and s-expressions
4. Data structures (suggestion)
5. General advice

Idea (recap)

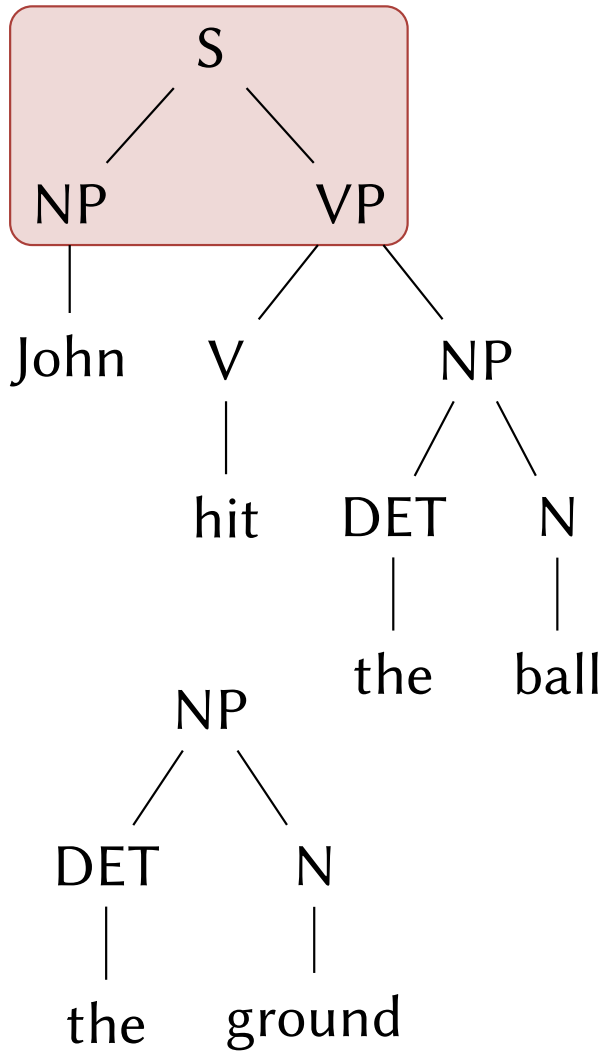
rules:



Idea (recap)

rules:

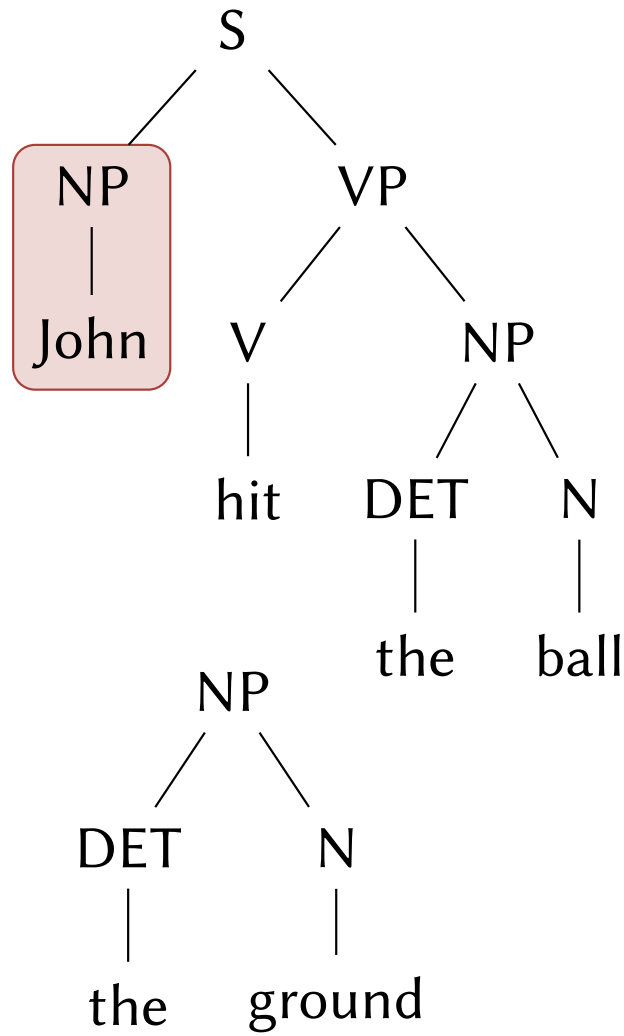
- $S \rightarrow NP\ VP$



Idea (recap)

rules:

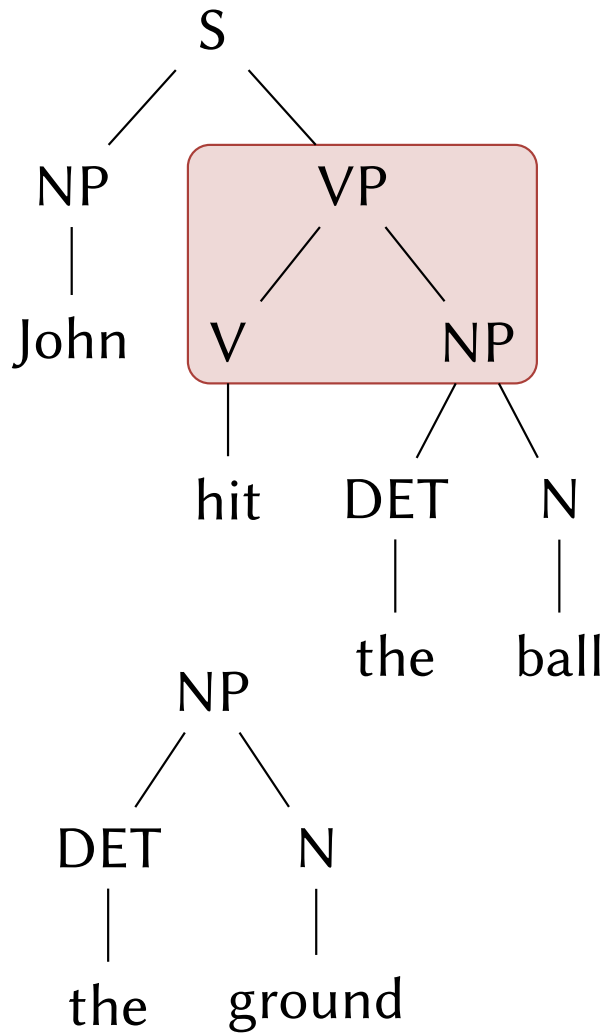
- $S \rightarrow NP\ VP$
- $NP \rightarrow John$



Idea (recap)

rules:

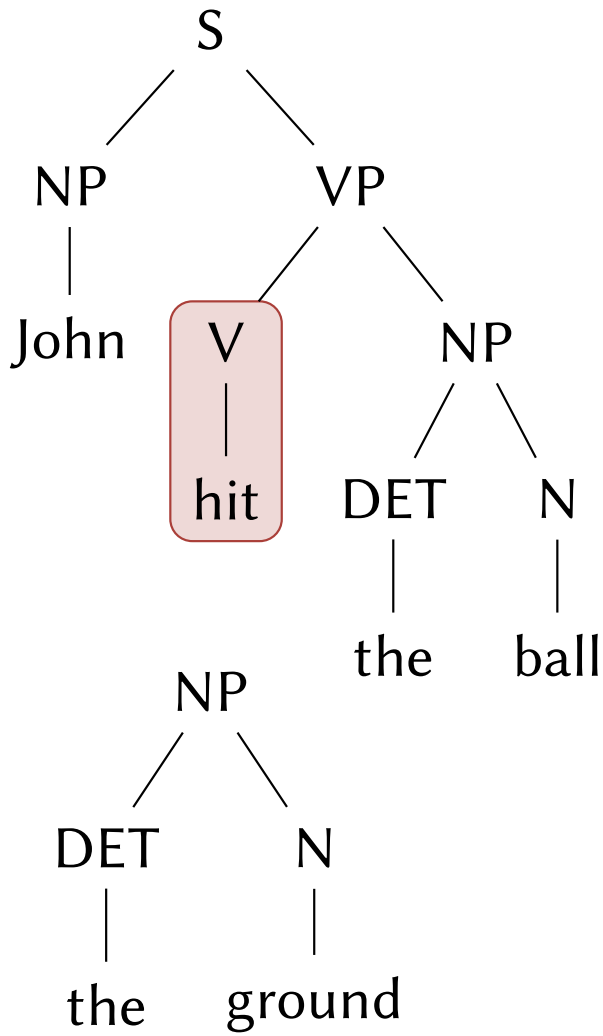
- S $\rightarrow$ NP VP
- NP $\rightarrow$ John
- VP $\rightarrow$ V NP



Idea (recap)

rules:

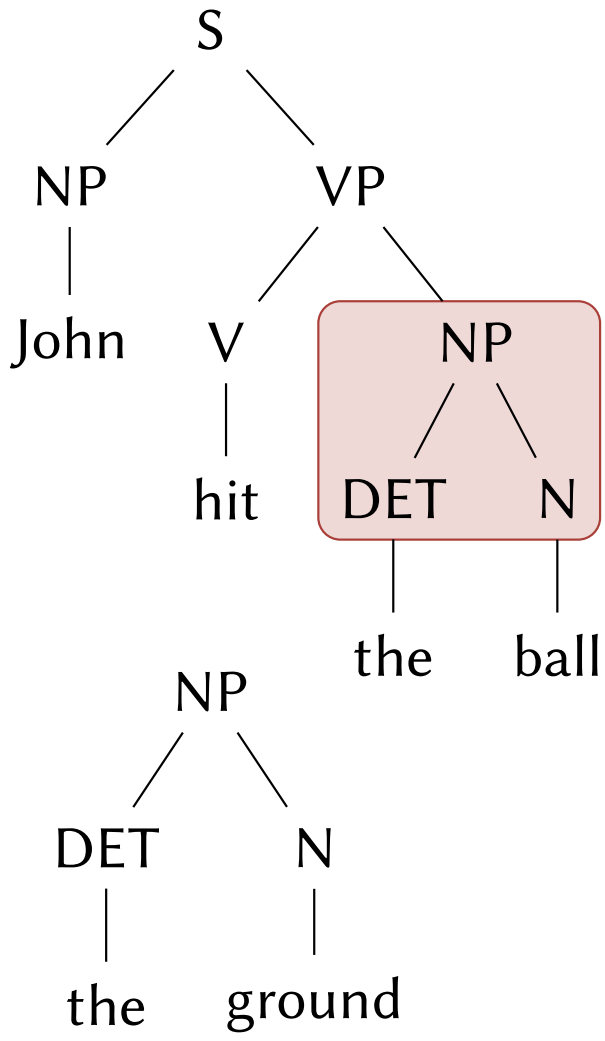
- S → NP VP
- NP → John
- VP → V NP
- V → hit



Idea (recap)

rules:

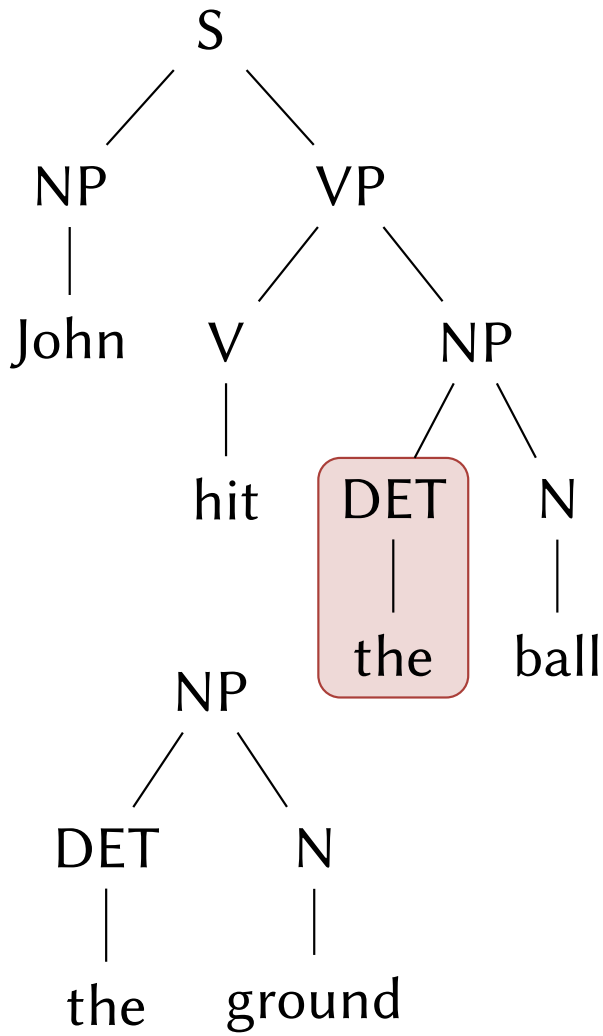
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N



Idea (recap)

rules:

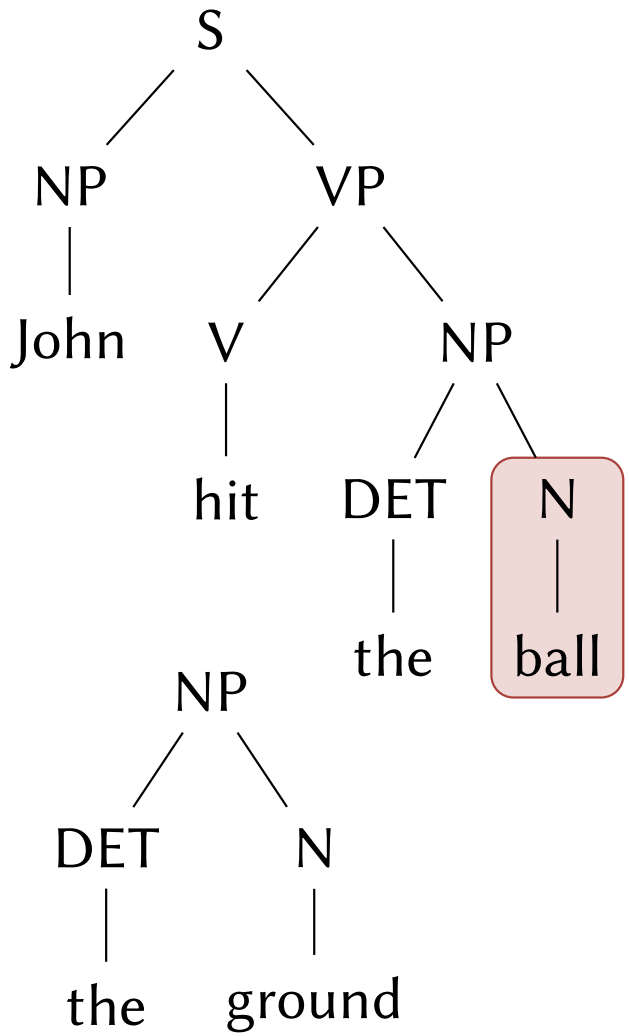
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N
- DET → the



Idea (recap)

rules:

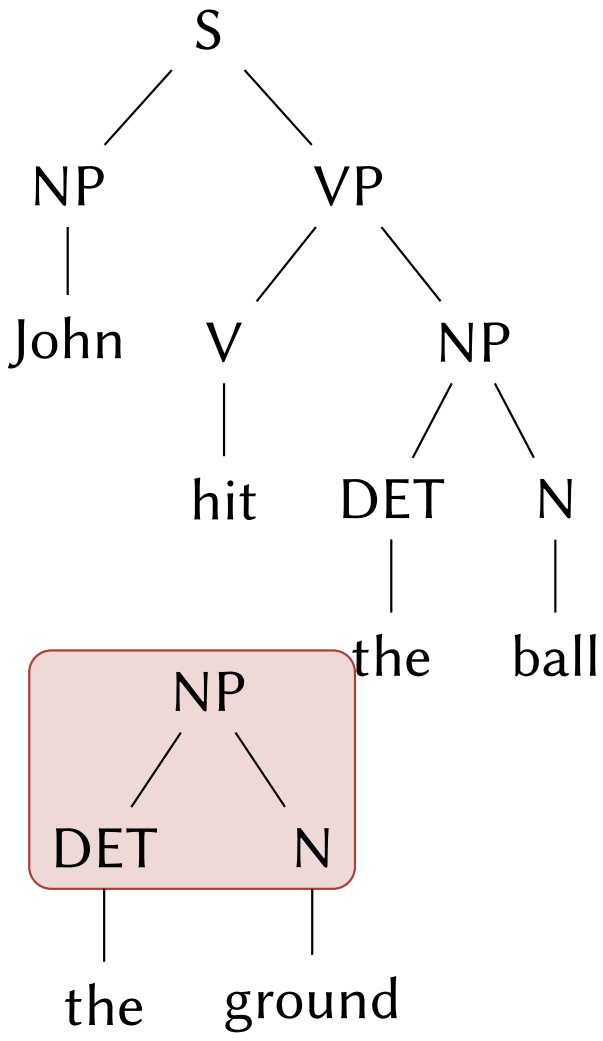
- S → NP VP
- NP → John
- VP → V NP
- V → hit
- NP → DET N
- DET → the
- N → ball



Idea (recap)

rules:

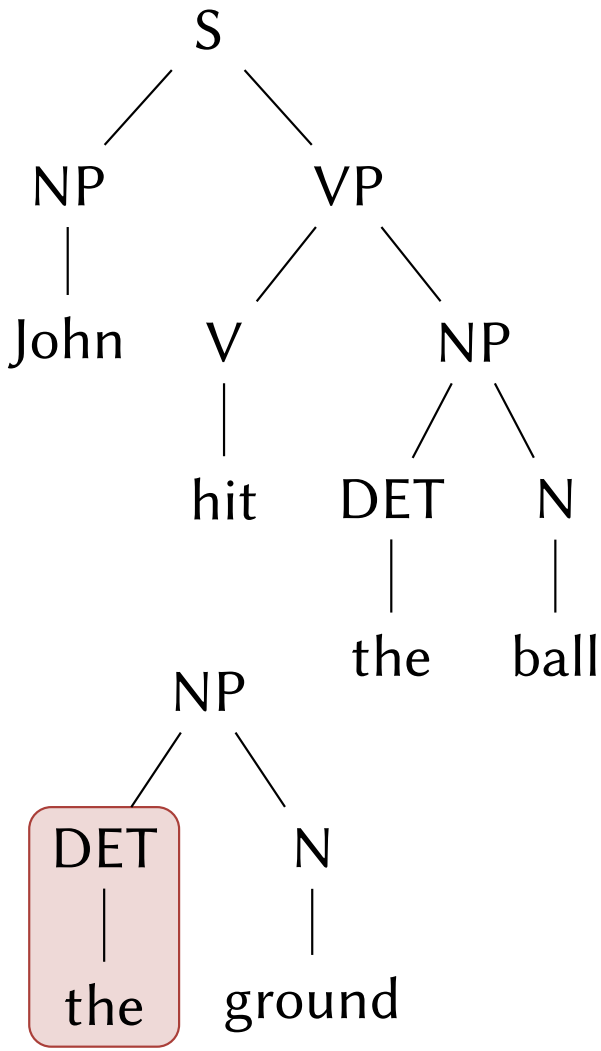
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 1
- N → ball # 1



Idea (recap)

rules:

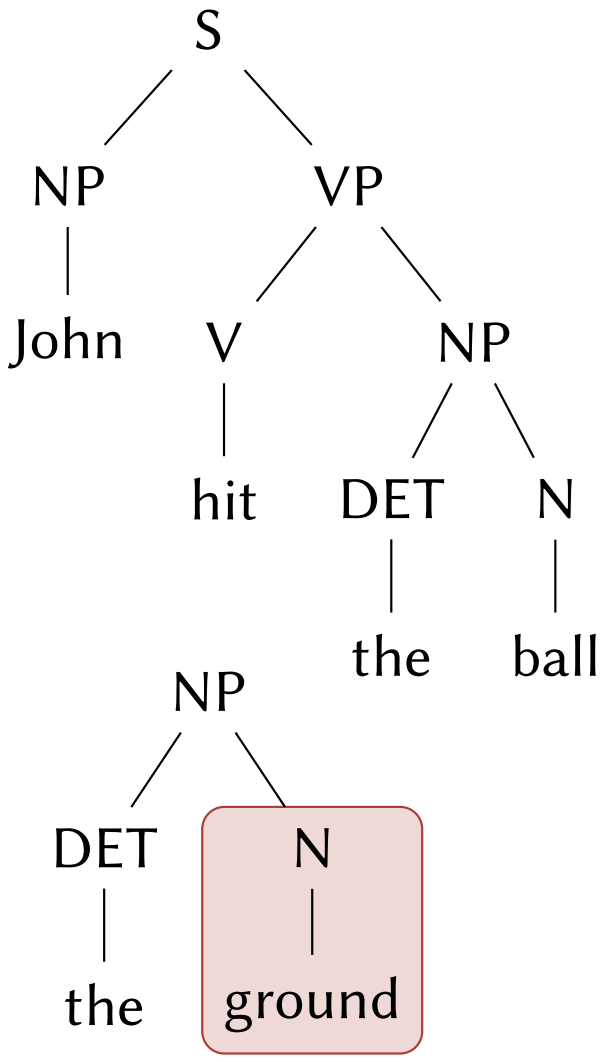
- S → NP VP # 1
- NP → John # 1
- VP → V NP # 1
- V → hit # 1
- NP → DET N # 2
- DET → the # 2
- N → ball # 1



Idea (recap)

rules:

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



Idea (recap)

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

Idea (recap)

rules:

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

normalised rules:

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

Idea (recap)

rules:

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

normalised rules:

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

Idea (recap)

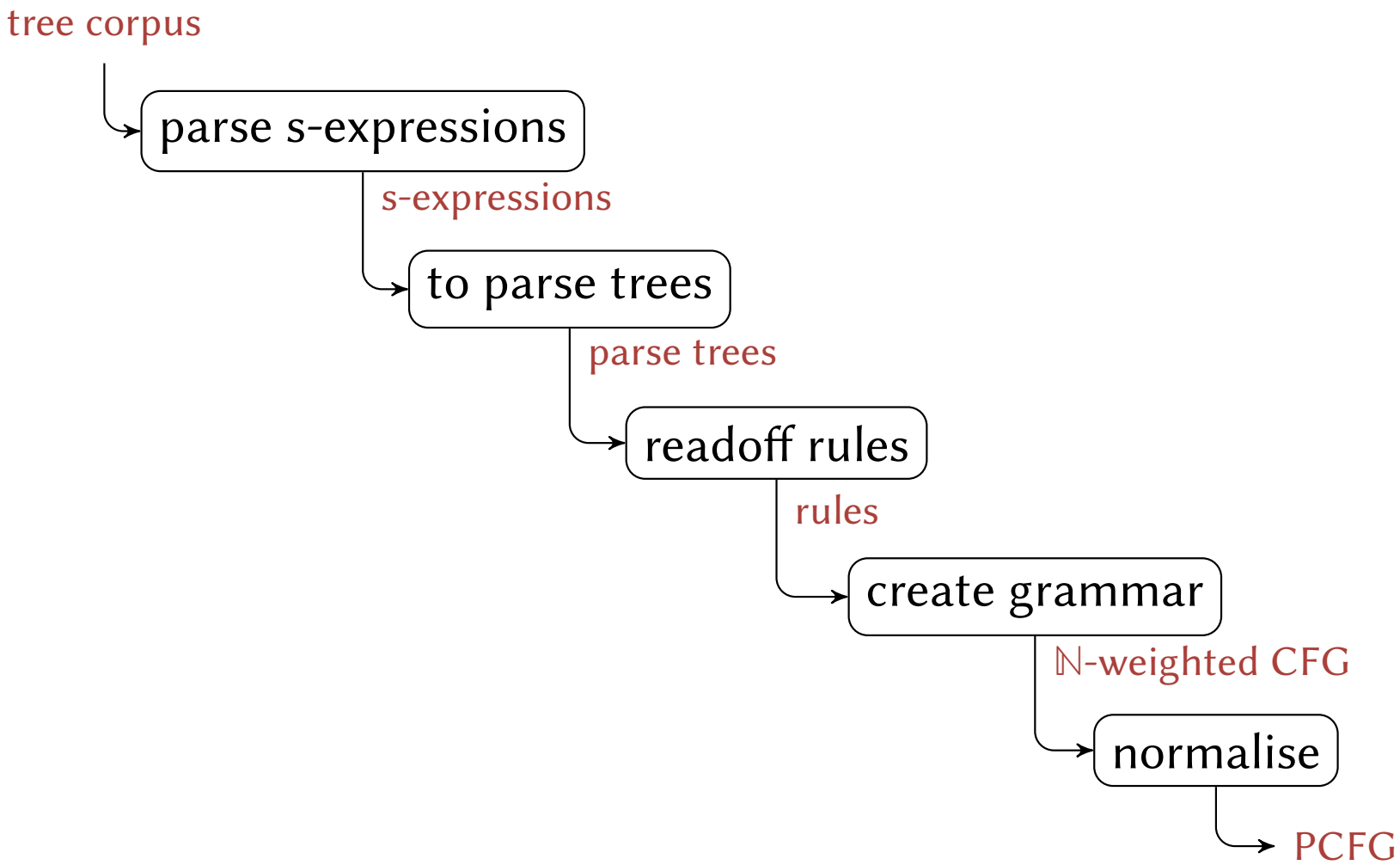
rules:

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

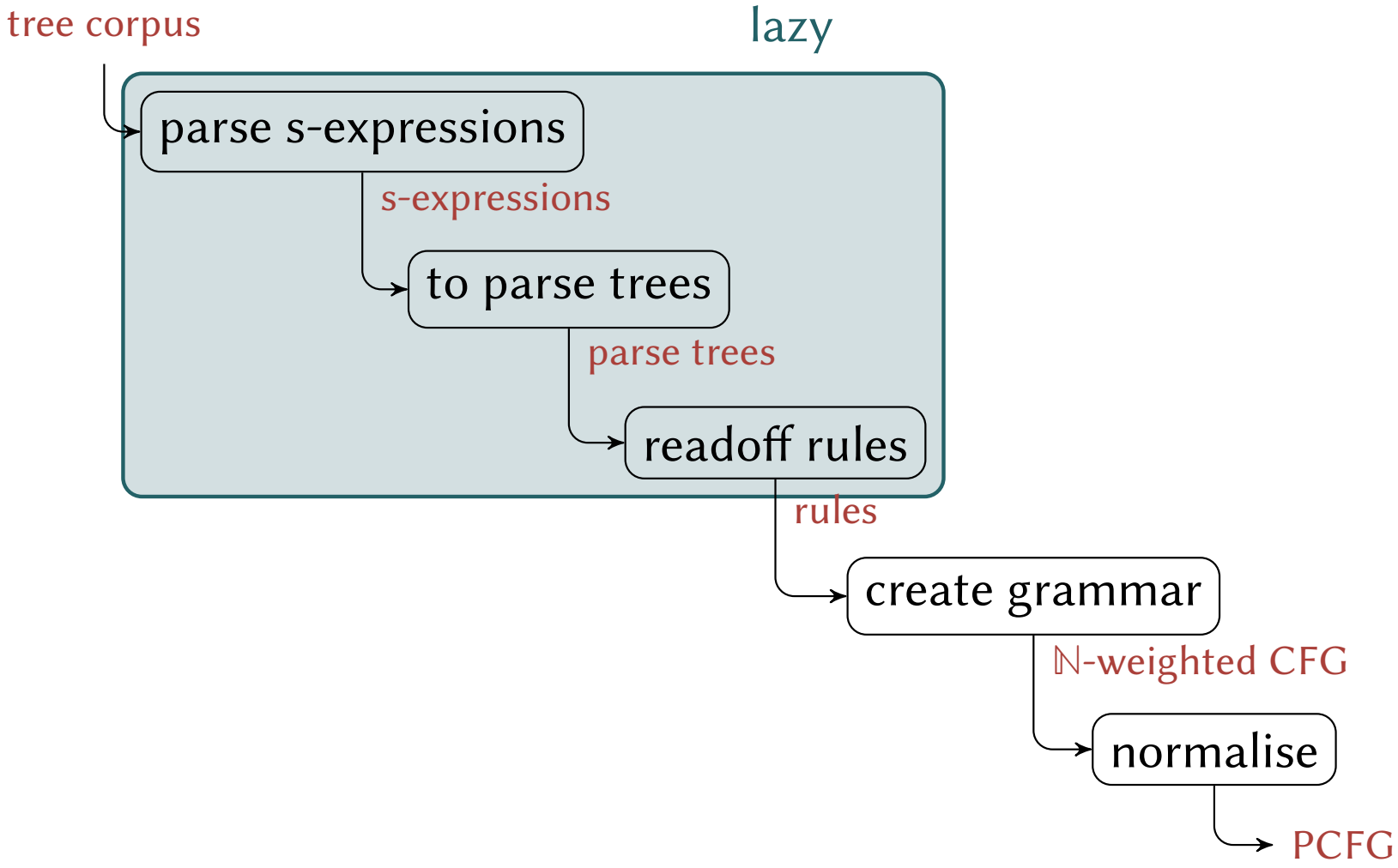
normalised rules:

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

Overall pipeline (suggestion)



Overall pipeline (suggestion)



Trees and s-expressions

- *s-expression over a set A:* symbolic **expressions**

  atoms: elements of A

  lists: strings $(s_1 \sqcup s_2 \sqcup \ldots \sqcup s_k)$ where the s_i's are s-expressions

Trees and s-expressions

- *s-expression over a set A:* symbolic **expressions**

  atoms: elements of A

  lists: strings $(s_1 \sqcup s_2 \sqcup \ldots \sqcup s_k)$ where the s_i's are s-expressions

- data structure in Rust (using tagged union):

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

Trees and s-expressions

- *s-expression over a set A:* symbolic **expressions**

  atoms: elements of A

  lists: strings $(s_1 \sqcup s_2 \sqcup \ldots \sqcup s_k)$ where the s_i's are s-expressions

- data structure in Rust (using tagged union):

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

- trees as s-expressions:
  - no empty lists
  - s_1 is always in A

Data structures (suggestion)

- tree

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

Data structures (suggestion)

- tree

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

- rule

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

Data structures (suggestion)

- tree

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

- 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>,
}
```

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)