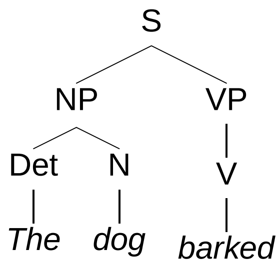


$$\begin{bmatrix} \texttt{LTOP} & h_1 \\ \texttt{INDEX} & e_2 \\ \texttt{RELS} & \left\langle \begin{bmatrix} \textit{prpstn\_m\_rel} \\ \texttt{LBL}\ h_1 \\ \texttt{MARG}\ h_3 \end{bmatrix} \begin{bmatrix} \textit{def\_q\_rel} \\ \texttt{LBL}\ h_4 \\ \texttt{ARG0}\ x_5 \\ \texttt{RSTR}\ h_6 \\ \texttt{BODY}\ h_7 \end{bmatrix} \begin{bmatrix} \textit{“dog\_n\_rel”} \\ \texttt{LBL}\ h_8 \\ \texttt{ARG0}\ x_5 \end{bmatrix} \begin{bmatrix} \textit{“bark\_v\_rel”} \\ \texttt{LBL}\ h_9 \\ \texttt{ARG0}\ e_2 \\ \texttt{ARG1}\ x_5 \end{bmatrix} \right\rangle \\ \texttt{HCONS} & \langle h_3 =_q h_9,\ h_6 =_q h_8 \rangle \end{bmatrix}$$

Computational Linguistics (INF2820 — Syntax)

$$S \longrightarrow NP\ VP;\ \ S \longrightarrow S\ PP;\ \ S \longrightarrow VP$$

Stephan Oepen

Universitetet i Oslo & CSLI Stanford

oe@ifi.uio.no

Candidate Theories of Grammar (1 of 3)

Language as a Set of Strings

The dog barks.
The angry dog barks.
The fierce dog barks.
The fierce angry dog barks.
The angry fierce dog barks.
The dog chased a cat.
A dog chased the cat.
The dog chased a black cat.
The dog chased a young cat.
The dog of my neighbours chased a cat.
A dog chased the cat of my neighbours.
The cat of my neighbours was chased by a dog.
...

Grammatical Categories (1 of 2)

Word Classes or Parts of Speech (PoS)

cat, *dog*, *neighbour(s)*, ...	**noun (N)**
adore, *bark(s)*, *chase(d)*, *was*, ...	**verb (V)**
fierce, *angry*, *black*, *young*, ...	**adjective (A)**
quickly, *probably*, *not*, ...	**adverb (Adv)**
a, *the*, *my*, *that*, ...	**determiner (D)**
of, *by*, *on*, *at*, *under*, ...	**preposition (P)**
she, *mine*, *those*, *what*, ...	**pronoun (Pro)**
and, *neither ... nor*, *because*, ...	**conjunction (C)**

the { *cat* / *dog* / **adore* }   *Kim likes to* { *bark* / *chase dogs* / **cat* }   *a* { *fierce* / *angry* / **quickly* } *cat*

Grammatical Categories (2 of 2)

Number — Person — Case — Gender

That dog barks. — Those dogs bark.

I bark. — You bark. — They bark. — Sam shaved himself.

We bark. — You bark. — Those dogs bark.

I saw her. — She saw me. — My dog barked.

...

How many distinct verb forms according to number and person?

Tense — Aspect — Mood

The dog barks. — The dog barked — The dog will bark.

The dog has barked. — The dog is barking.

If I were a carpenter, ...

...

Candidate Theories of Grammar (2 of 3)

Language as a Sequence of Word Classes

cat, *dog*, *neighbour(s)*, ...	**noun (N)**
adore, *bark(s)*, *chase(d)*, *was*, ...	**verb (V)**
fierce, *angry*, *black*, *young*, ...	**adjective (A)**
a, *the*, *my*, *that*, ...	**determiner (D)**
of, *by*, *on*, *at*, *under*, ...	**preposition (P)**

Regular Expressions

$$\mathsf{D}^{?}\ \mathsf{A}^{*}\ \mathsf{N}^{+}\ \mathsf{V}\ \left(\mathsf{D}^{?}\ \mathsf{A}^{*}\ \mathsf{N}^{+}\right)^{*}$$

Candidate Theories of Grammar (2 of 3)

Language as a Sequence of Word Classes

cat, *dog*, *neighbour(s)*, ...	**noun (N)**
adore, *bark(s)*, *chase(d)*, *was*, ...	**verb (V)**
fierce, *angry*, *black*, *young*, ...	**adjective (A)**
a, *the*, *my*, *that*, ...	**determiner (D)**
of, *by*, *on*, *at*, *under*, ...	**preposition (P)**

Regular Expressions

$$\mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\ \mathrm{V}\ \left(\mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\right)^{*}$$

$$\mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\ \left(\mathrm{P}\ \mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\right)^{*}\ \mathrm{V}\ \left(\mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\ \left(\mathrm{P}\ \mathrm{D}^{?}\ \mathrm{A}^{*}\ \mathrm{N}^{+}\right)^{*}\ \right)^{*}$$

Candidate Theories of Grammar (3 of 3)

Mildly Mathematically: Context-Free Grammars

- Formally, a *context-free grammar* (CFG) is a quadruple: $\langle C, \Sigma, P, S \rangle$
- C is the set of categories (aka *non-terminals*), e.g. $\{$S, NP, VP, V$\}$;
- Σ is the vocabulary (aka *terminals*), e.g. $\{$Kim, snow, saw, in$\}$;
- P is a set of category rewrite rules (aka *productions*), e.g.

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
NP $\rightarrow$ Kim
NP $\rightarrow$ snow
V $\rightarrow$ saw

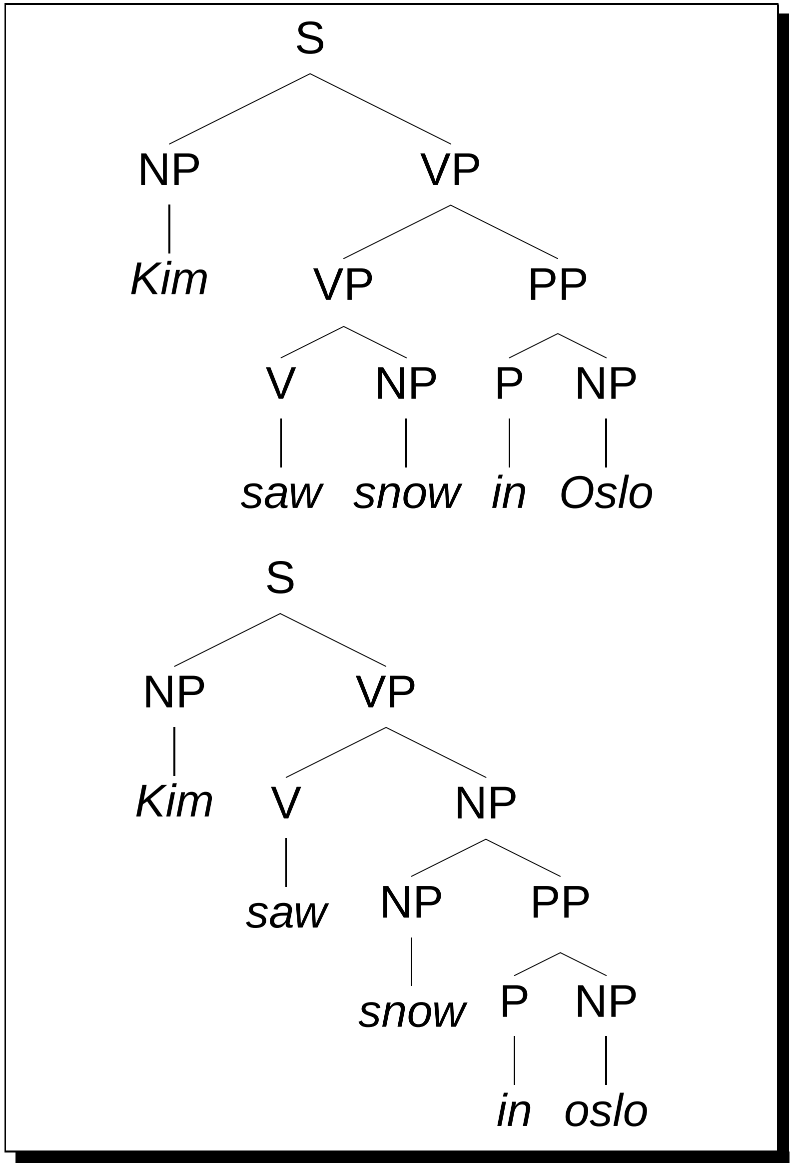
- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Parsing: Recognizing the Language of a Grammar

S $\longrightarrow$ NP VP
VP $\longrightarrow$ V NP
VP $\longrightarrow$ VP PP
NP $\longrightarrow$ NP PP
PP $\longrightarrow$ P NP
NP $\longrightarrow$ Kim | snow | Oslo
V $\longrightarrow$ saw
P $\longrightarrow$ in

All Complete Derivations

- are rooted in the start symbol S;
- label internal nodes with categories $\in C$, leafs with words $\in \Sigma$;
- instantiate a grammar rule $\in P$ at each local subtree of depth one.



Limitations of Context-Free Grammar

Agreement and Valency (For Example)

That dog barks.

**That dogs barks.*

**Those dogs barks.*

The dog chased a cat.

**The dog barked a cat.*

**The dog chased.*

**The dog chased a cat my neighbours.*

The cat was chased by a dog.

**The cat was chased of a dog.*

...

A Simple-Minded Parsing Algorithm

Control Structure

- top-down: given a parsing goal α, use all grammar rules that rewrite α;
- successively instantiate (extend) the right-hand sides of each rule;
- for each β_i in the RHS of each rule, recursively attempt to parse β_i;
- termination: when α is a prefix of the input string, parsing succeeds.

(Intermediate) Results

- Each result records a (partial) tree and remaining input to be parsed;
- complete results consume the full input string and are rooted in S;
- whenever a RHS is fully instantiated, a new tree is built and returned;
- all results at each level are combined and successively accumulated.

The Recursive Descent Parser

```
(defun parse (input goal)
  (if (equal (first input) goal)
    (let ((edge (make-edge :category (first input))))
      (list (make-parse :edge edge :input (rest input))))
    (loop
        for rule in (rules-deriving goal)
        append (extend-parse (rule-lhs rule) nil (rule-rhs rule) input))))
```

```
(defun extend-parse (goal analyzed unanalyzed input)
  (if (null unanalyzed)
    (let ((edge (make-edge :category goal :daughters analyzed)))
      (list (make-parse :edge edge :input input)))
    (loop
        for parse in (parse input (first unanalyzed))
        append (extend-parse
                goal (append analyzed (list (parse-edge parse)))
                (rest unanalyzed)
                (parse-input parse)))))
```