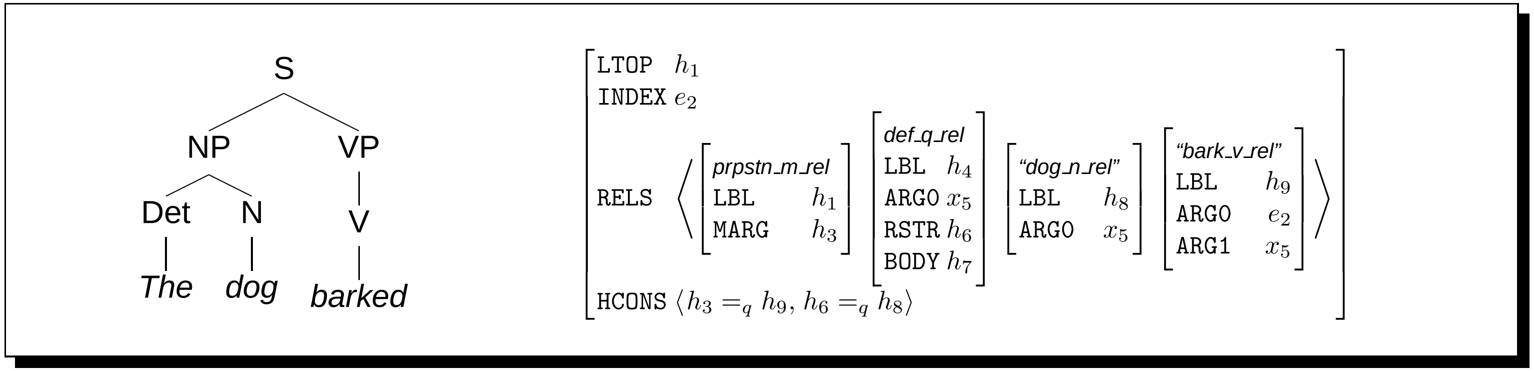


Algorithms for AI and NLP (INF4820 — Parsing)

$S \longrightarrow NP\ VP;\ \ NP \longrightarrow Det\ N;\ \ VP \longrightarrow V\ NP$

Stephan Oepen and Jonathon Read

Universitetet i Oslo

{ oe | jread }@ifi.uio.no

A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

Meaning

- *Kim gave Sandy the book.*
- *Kim gave the book to Sandy.*
- *Sandy was given the book by Kim.*

A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

Meaning

- *Kim gave Sandy the book.*
- *Kim gave the book to Sandy.*
- *Sandy was given the book by Kim.*

Ambiguity

- *Kim saw the astronomer with the telescope.*
- *Have her report on my desk by Friday!*

A Grossly Simplified Example

The Grammar of Spanish

$S \longrightarrow$ NP VP

$VP \longrightarrow$ V NP

$VP \longrightarrow$ VP PP

$PP \longrightarrow$ P NP

$NP \longrightarrow$ “nieve”

$NP \longrightarrow$ “Juan”

$NP \longrightarrow$ “Oslo”

$V \longrightarrow$ “amó”

$P \longrightarrow$ “en”

Juan amó nieve en Oslo

A Grossly Simplified Example

The Grammar of Spanish

S → NP VP

VP → V NP

VP → VP PP

PP → P NP

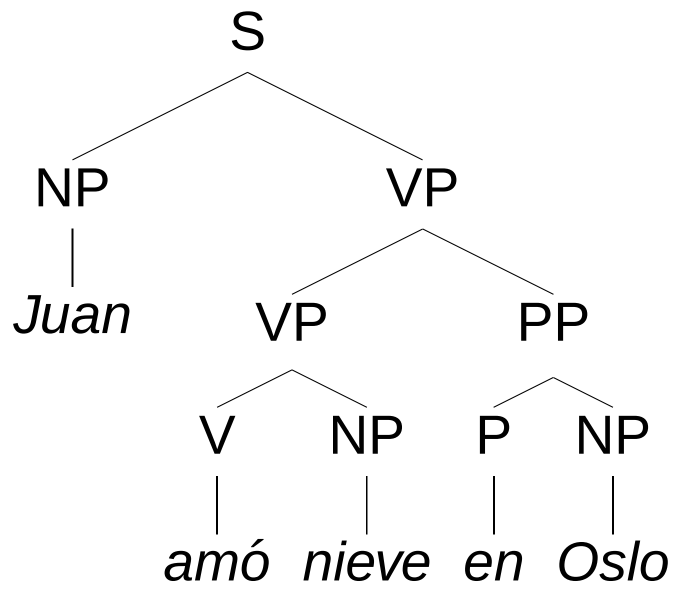
NP → "nieve"

NP → "Juan"

NP → "Oslo"

V → "amó"

P → "en"

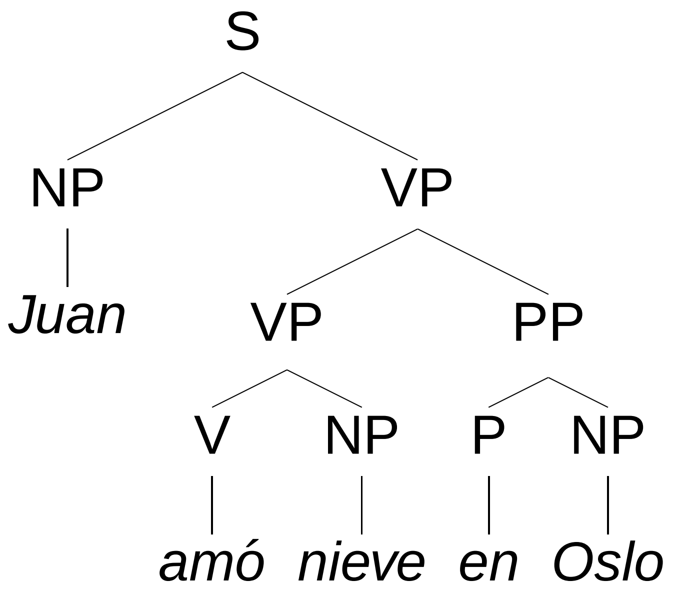


Juan amó nieve en Oslo

A Grossly Simplified Example

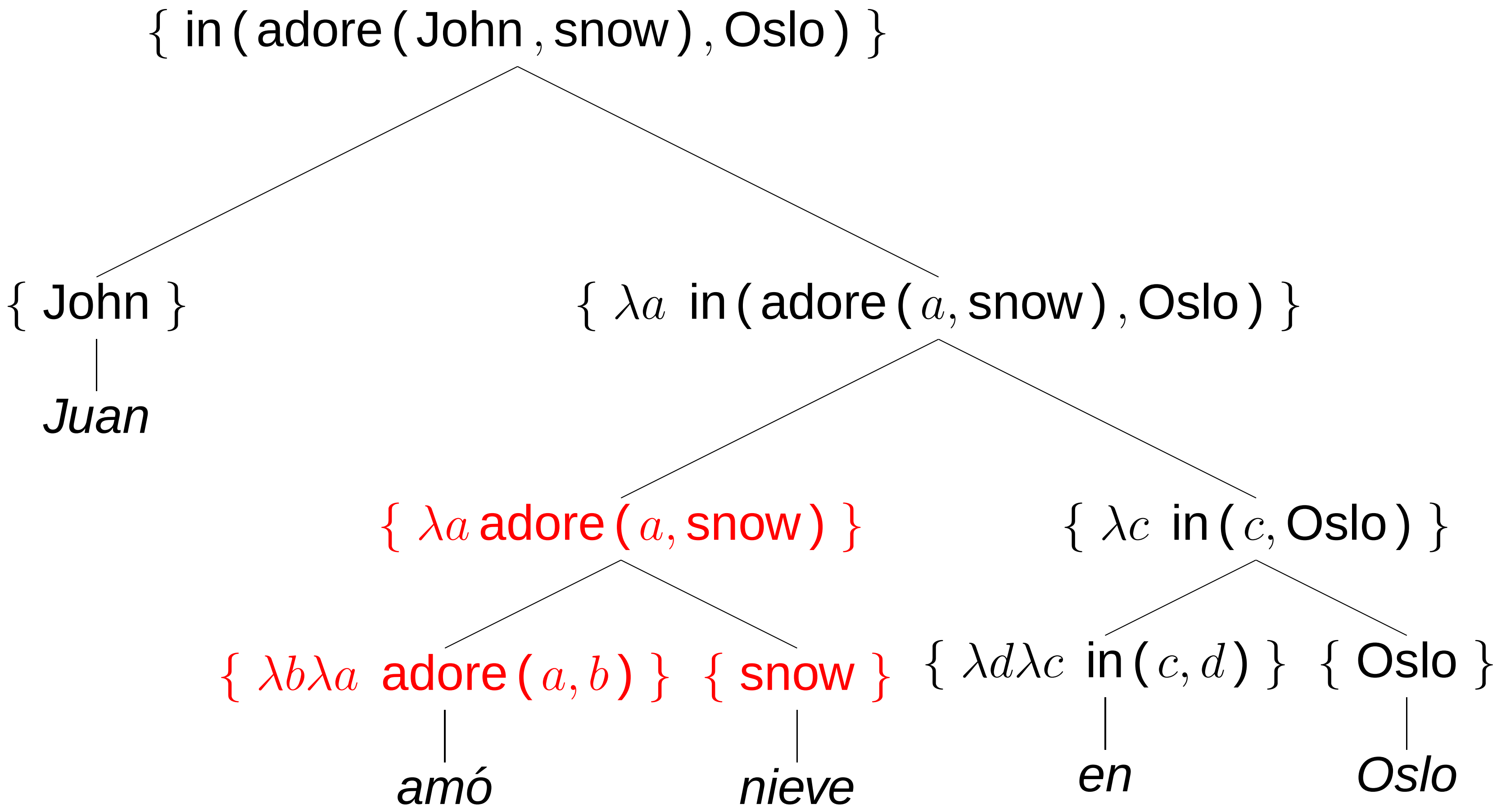
The Grammar of Spanish

S $\rightarrow$ NP VP	{ VP (NP) }
VP $\rightarrow$ V NP	{ V (NP) }
VP $\rightarrow$ VP PP	{ PP (VP) }
PP $\rightarrow$ P NP	{ P (NP) }
NP $\rightarrow$ “nieve”	{ snow }
NP $\rightarrow$ “Juan”	{ John }
NP $\rightarrow$ “Oslo”	{ Oslo }
V $\rightarrow$ “amó”	{ $\lambda b \lambda a$ adore (a, b) }
P $\rightarrow$ “en”	{ $\lambda d \lambda c$ in (c, d) }



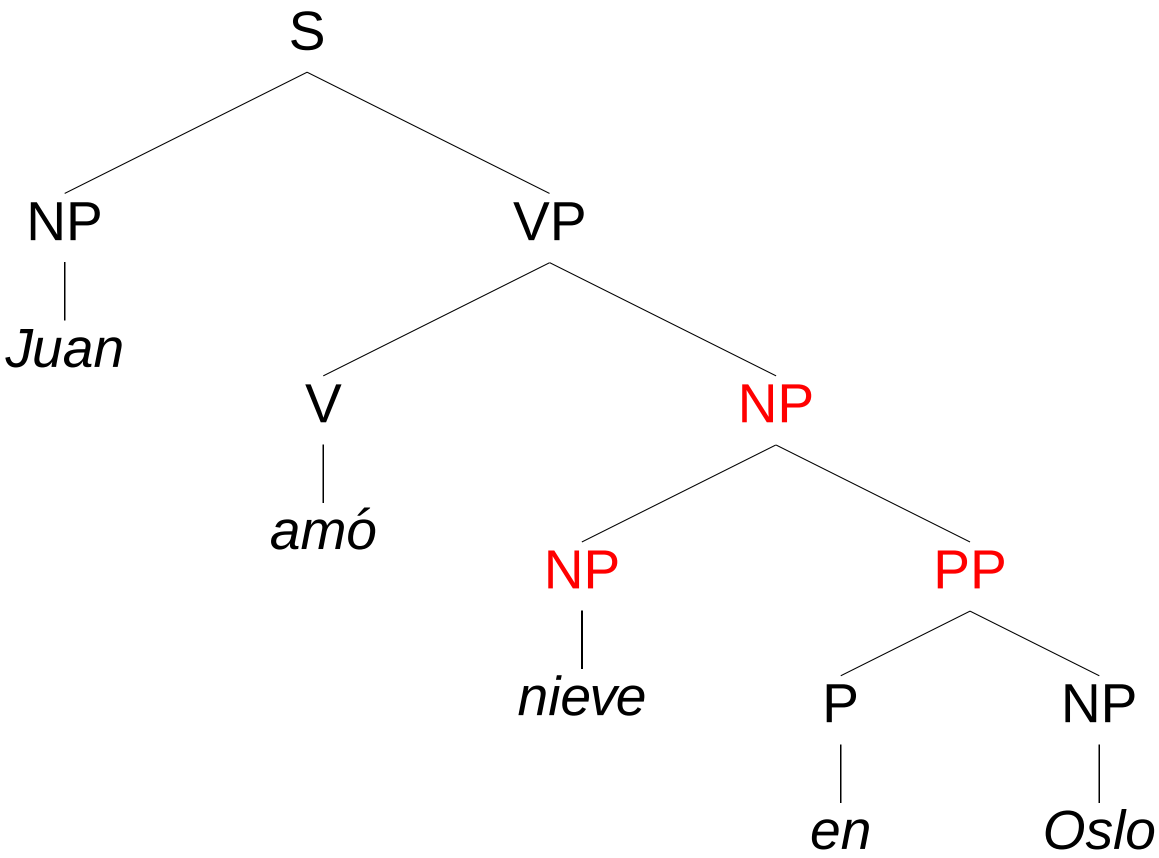
Juan amó nieve en Oslo

Meaning Composition (Grossly Simplified, Still)



VP $\rightarrow$ V NP { V (NP) }

Another Interpretation — Structural Ambiguity



$\text{NP} \rightarrow \text{NP PP}$ { PP (NP) }

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. $\{\mathsf{S}, \mathsf{NP}, \mathsf{VP}, \mathsf{V}\}$;
- Σ is the vocabulary (aka *terminals*), e.g. $\{\mathsf{Kim}, \mathsf{snow}, \mathsf{saw}, \mathsf{in}\}$;
- 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

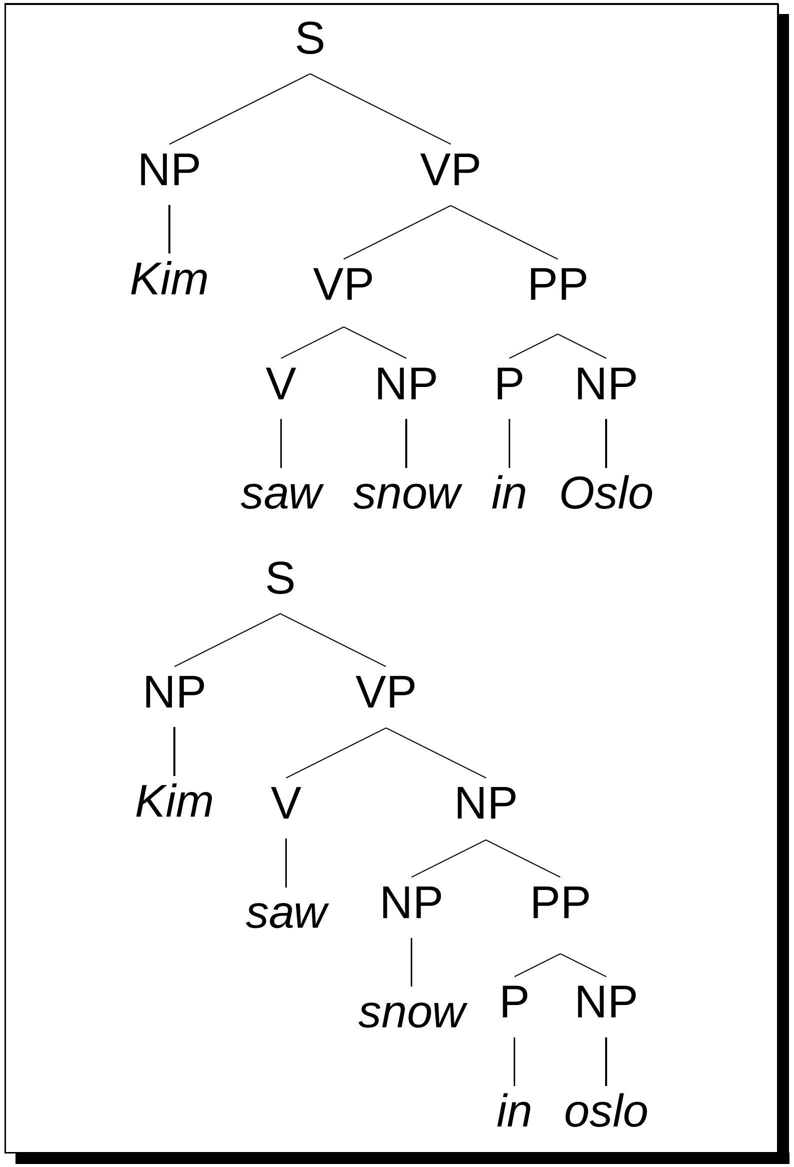
- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Parsing: Recognizing the Language of a Grammar

S $\rightarrow$ NP VP
VP $\rightarrow$ V | V NP | VP PP
NP $\rightarrow$ NP PP
PP $\rightarrow$ P NP
NP $\rightarrow$ Kim | snow | Oslo
V $\rightarrow$ saw
P $\rightarrow$ 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.



Recursive Descend: A Naïve 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

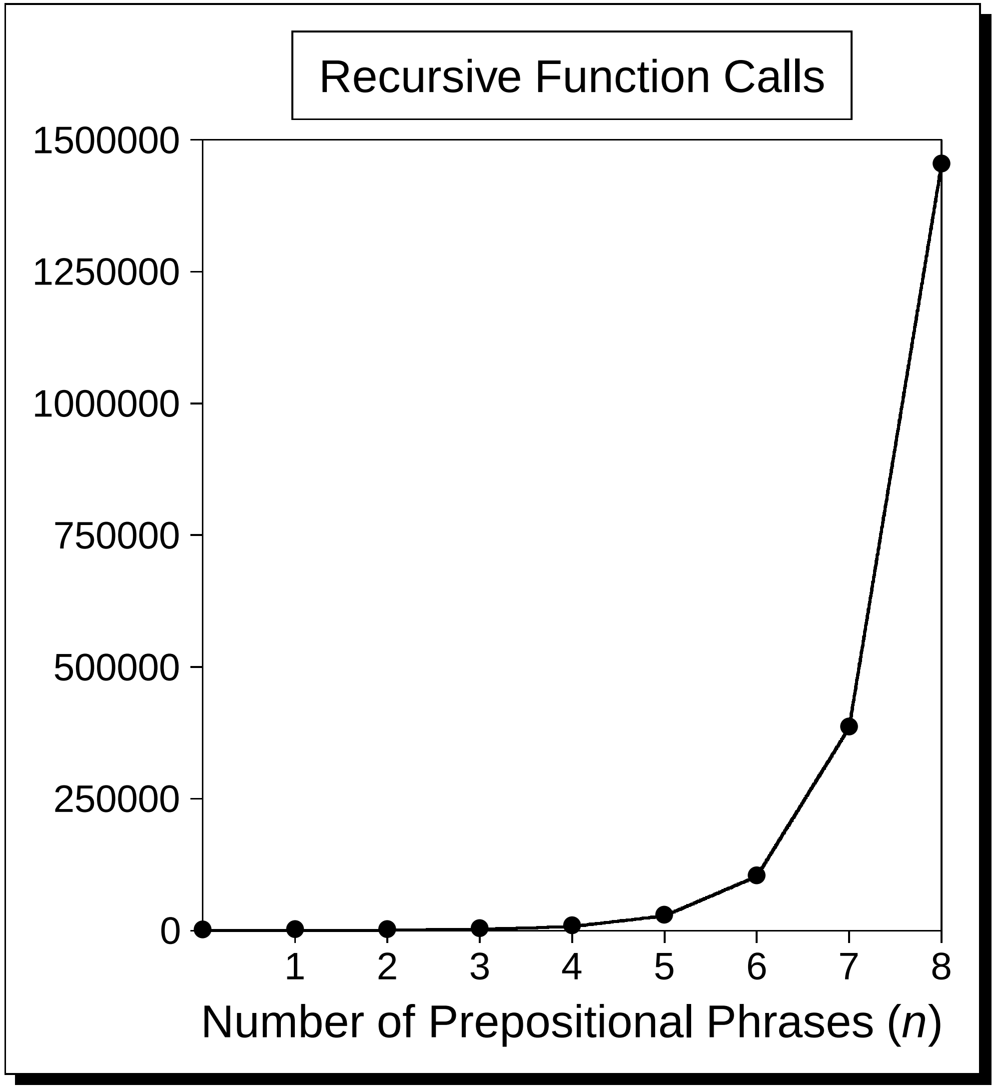
- Each result records a (partial) tree and remaining input to be parsed;
- complete results consume the full input string and are rooted in S;
- whenever a RHS is fully instantiated, a new tree is built and returned;
- all results at each level are combined and successively accumulated.

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)))))
```

Quantifying the Complexity of the Parsing Task



Kim adores snow (in Oslo)^n

n	trees	calls
0	1	46
1	2	170
2	5	593
3	14	2,093
4	42	7,539
5	132	27,627
6	429	102,570
7	1430	384,566
8	4862	1,452,776
⋮	⋮	⋮

Top-Down vs. Bottom-Up Parsing

Top-Down (Goal-Oriented)

- Left recursion (e.g. a rule like ‘VP → VP PP’) causes infinite recursion;
- grammar conversion techniques (eliminating left recursion) exist, but will typically be undesirable for natural language processing applications;

→ assume bottom-up as basic search strategy for remainder of the course.

Bottom-Up (Data-Oriented)

- unary (left-recursive) rules (e.g. ‘NP → NP’) would still be problematic;
- lack of parsing goal: compute all possible derivations for, say, the input *adores snow*; however, it is ultimately rejected since it is not sentential;
- availability of partial analyses desirable for, at least, some applications.

Chart Parsing — Specialized Dynamic Programming

Basic Notions

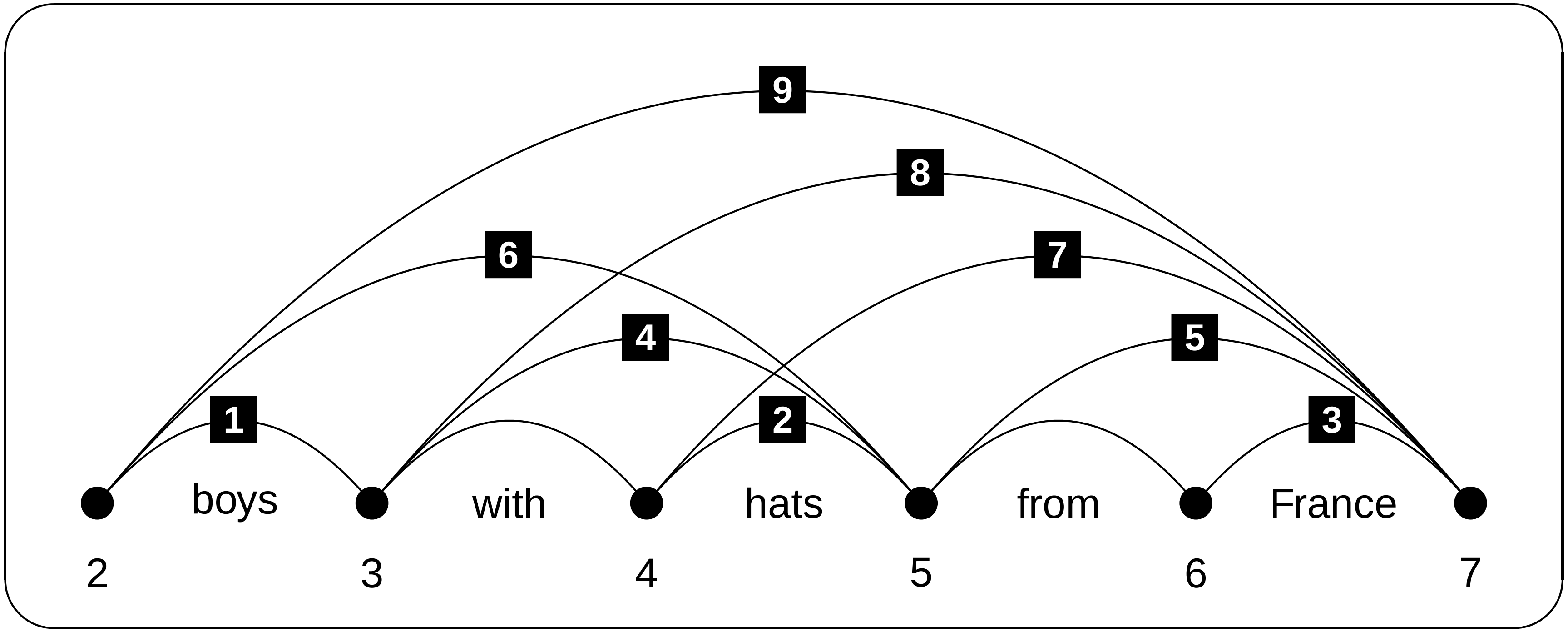
- Use *chart* to record partial analyses, indexing them by string positions;
- count inter-word vertices; CKY: chart row is *start*, column *end* vertex;
- treat multiple ways of deriving the same category for some substring as *equivalent*; pursue only once when combining with other constituents.

Key Benefits

- Dynamic programming (memoization): avoid recomputation of results;
- efficient indexing of constituents: no search by start or end positions;
- compute *parse forest* with exponential 'extension' in *polynomial* time.

Bounding Ambiguity — The Parse Chart

- For many substrings, more than one way of deriving the same category;
- NPs: 1 | 2 | 3 | 6 | 7 | 9; PPs: 4 | 5 | 8; 9 ≡ 1 + 8 | 6 + 5;
- *parse forest* — a single item represents multiple trees [Billot & Lang, 89].



The CKY (Cocke, Kasami, & Younger) Algorithm

for ($0 \leq i < |\textit{input}|$) do
  $\textit{chart}_{[i,i+1]} \leftarrow \{\alpha \mid \alpha \rightarrow \textit{input}_i \in P\}$;
for ($1 \leq l < |\textit{input}|$) do
  for ($0 \leq i < |\textit{input}| - l$) do
    for ($1 \leq j \leq l$) do
      if ($\alpha \rightarrow \beta_1 \, \beta_2 \in P \wedge \beta_1 \in \textit{chart}_{[i,i+j]} \wedge \beta_2 \in \textit{chart}_{[i+j,i+l+1]}$) then
        $\textit{chart}_{[i,i+l+1]} \leftarrow \textit{chart}_{[i,i+l+1]} \cup \{\alpha\}$;

$[0,2] \leftarrow [0,1] + [1,2]$
$\ldots$
$[0,5] \leftarrow [0,1] + [1,5]$
$[0,5] \leftarrow [0,2] + [2,5]$
$[0,5] \leftarrow [0,3] + [3,5]$
$[0,5] \leftarrow [0,4] + [4,5]$

	1	2	3	4	5
0	NP		S		S
1		V	VP		VP
2			NP		NP
3				P	PP
4					NP

Limitations of the CKY Algorithm

Built-In Assumptions

- *Chomsky Normal Form* grammars: $\alpha \rightarrow \beta_1\beta_2$ or $\alpha \rightarrow \gamma$ ($\beta_i \in C$, $\gamma \in \Sigma$);
- breadth-first (aka exhaustive): always compute all values for each cell;
- rigid control structure: bottom-up, left-to-right (one diagonal at a time).

Generalized Chart Parsing

- Liberate order of computation: no assumptions about earlier results;
- *active edges* encode partial rule instantiations, 'waiting' for additional (adjacent and passive) constituents to complete: $[1, 2, \text{VP} \rightarrow \text{V} \bullet \text{NP}]$;
- parser can fill in chart cells in *any* order and guarantee completeness.