Computational Linguistics (INF2820 — Parsing)

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

Stephan Oepen

Universitetet i Oslo & CSLI Stanford

oe@ifi.uio.no

Background: Trees as Bracketed Sequences

- Trees can be encoded as sequences (*dominance* plus *precedence*):

```
(S (NP kim)
   (VP (V adored)
       (NP snow)))
```
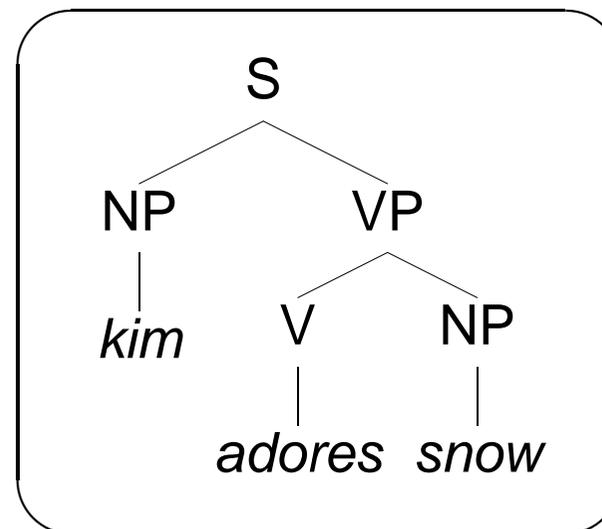

- the `first()` element (at each level) represents the tree root (or mother);

- all other elements (i.e. the `rest()`) correspond to immediate daughters.

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

```lisp
(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)))) 
```

```lisp
(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))))) 
```

A Closer Look at the Calling Sequence

```
SSP(18): (parse '(kim adored snow) 's)
parse(): input: (KIM ADORED SNOW); goal: S
  parse(): input: (KIM ADORED SNOW); goal: NP
    parse(): input: (KIM ADORED SNOW); goal: KIM
    parse(): input: (KIM ADORED SNOW); goal: SANDY
    parse(): input: (KIM ADORED SNOW); goal: SNOW
  parse(): input: (ADORED SNOW); goal: VP
    parse(): input: (ADORED SNOW); goal: V
      parse(): input: (ADORED SNOW); goal: LAUGHED
      parse(): input: (ADORED SNOW); goal: ADORED
    parse(): input: (ADORED SNOW); goal: V
      parse(): input: (ADORED SNOW); goal: LAUGHED
      parse(): input: (ADORED SNOW); goal: ADORED
    parse(): input: (SNOW); goal: NP
      ...
```

Natural Language Understanding (5)

Quantifying the Complexity of the Parsing Task

Recursive Function Calls

Number of Prepositional Phrases (n)
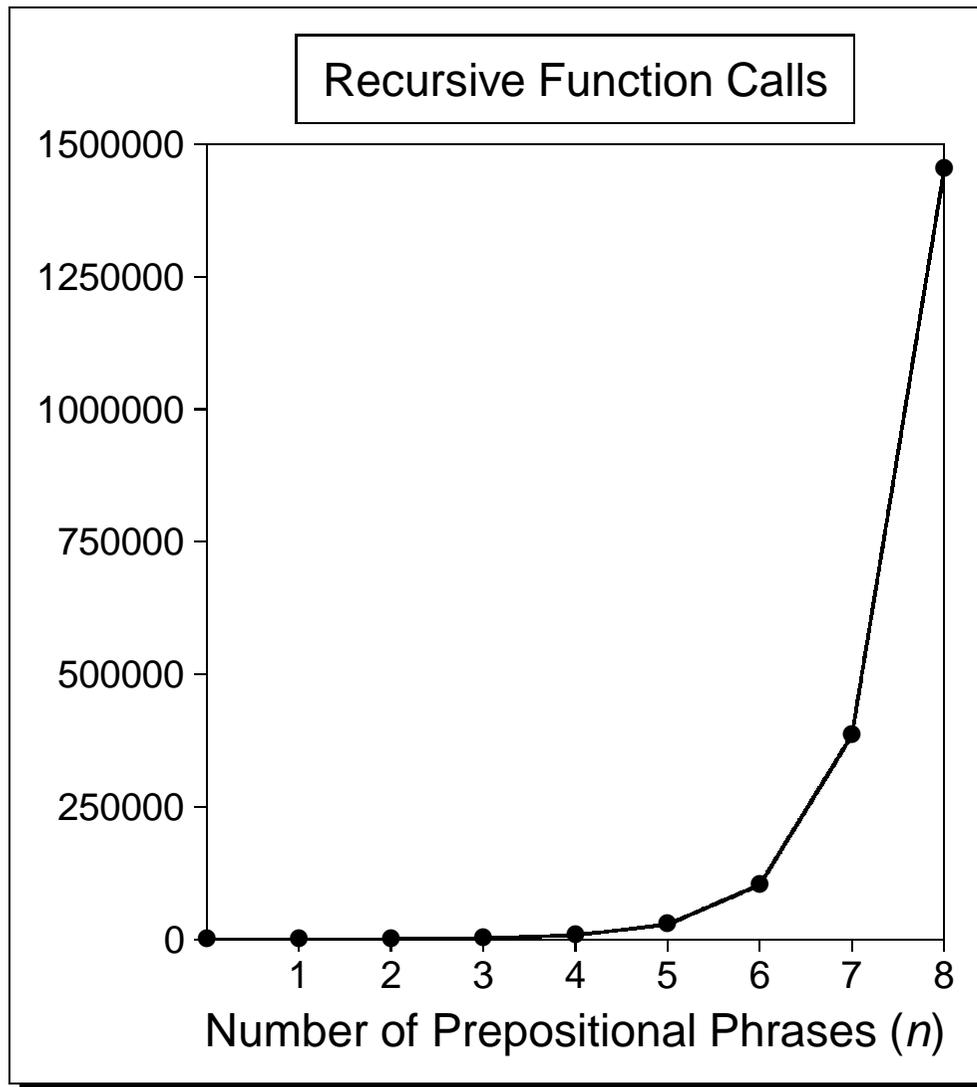
Kim adores snow (*in Oslo*)^n

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
⋮	⋮	⋮

Memoization: Remember Earlier Results

Dynamic Programming

- The function call `(parse (adored snow) V)` executes two times;

- *memoization* — record `parse()` results for each set of arguments;

$\rightarrow$ requires abstract data type, efficient indexing on *input* and *goal*.
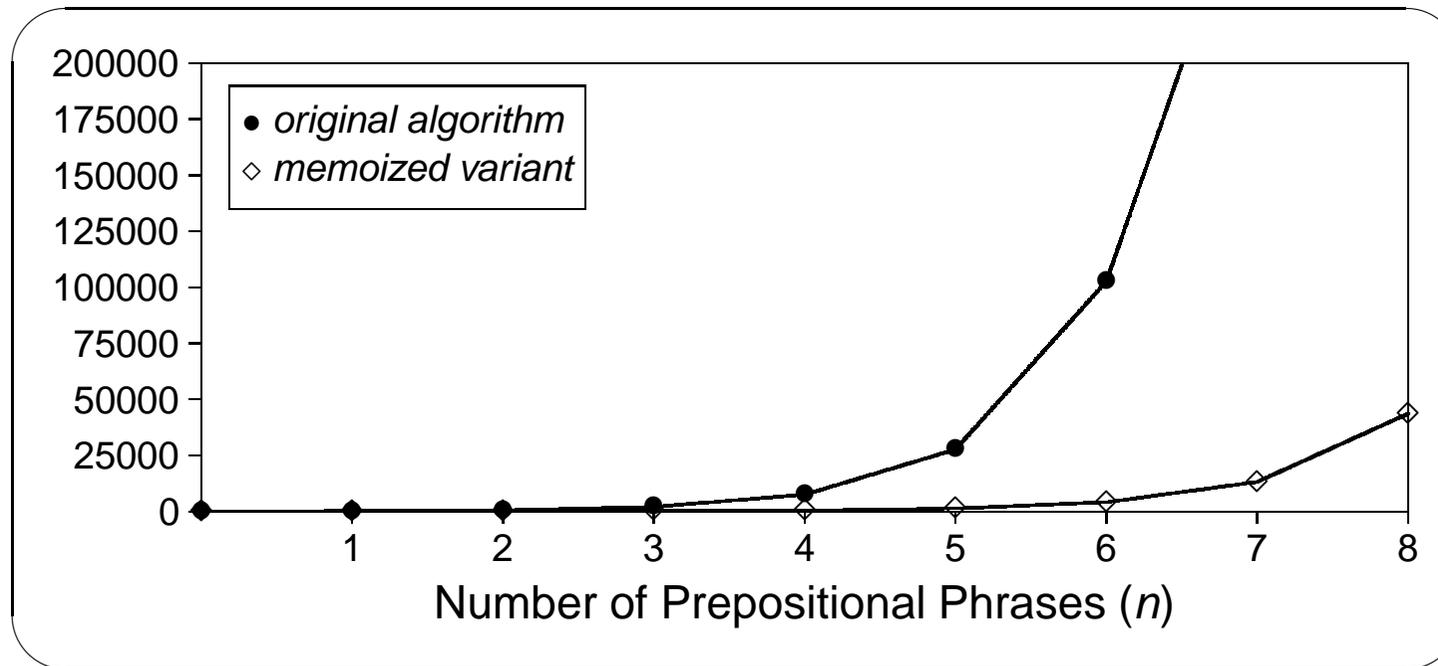
Top-Down vs. Bottom-Up Parsing

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].
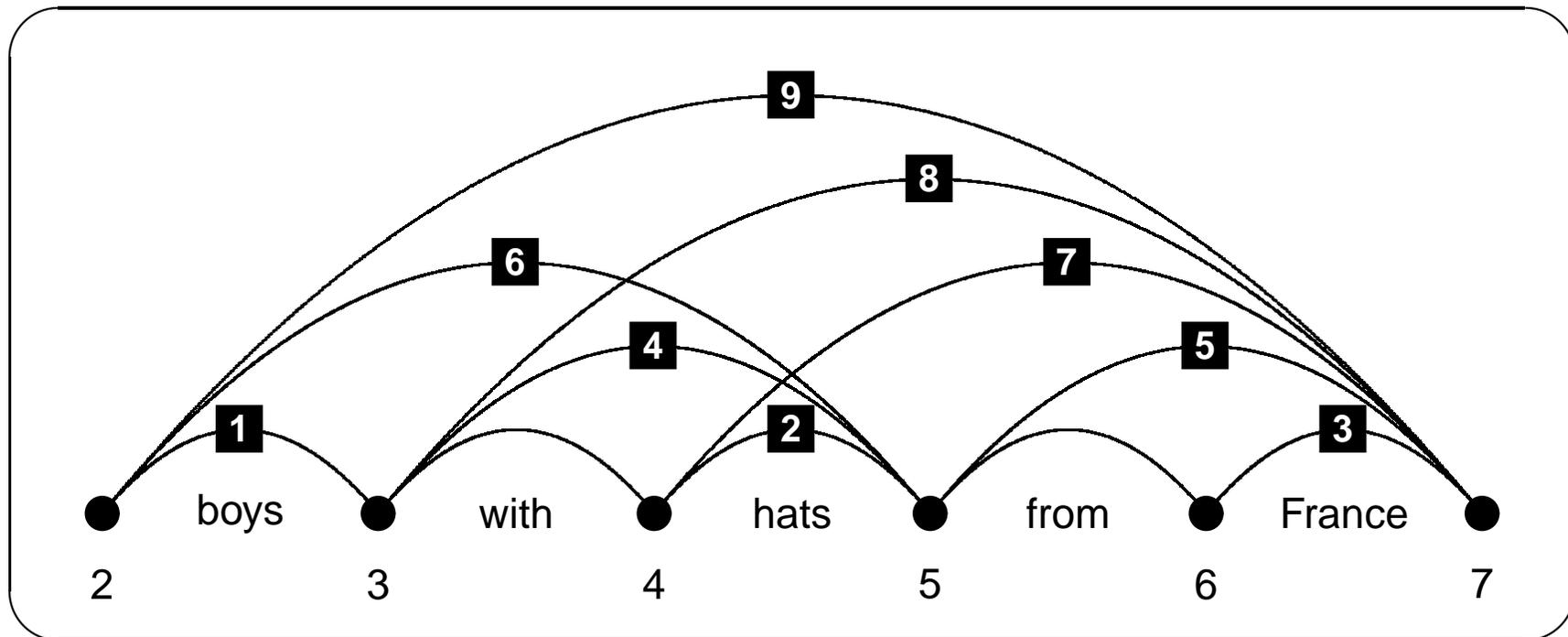
The CKY (Cocke, Kasami, & Younger) Algorithm

for ($0 \leq i < |input|$) do
  $chart_{[i,i+1]} \leftarrow \{\alpha \,|\, \alpha \rightarrow input_i \in P\}$;
for ($1 \leq l < |input|$) do
  for ($0 \leq i < |input| - l$) do
    for ($1 \leq j \leq l$) do
      if ($\alpha \rightarrow \beta_1\,\beta_2 \in P \land \beta_1 \in chart_{[i,i+j]} \land \beta_2 \in chart_{[i+j,i+l+1]}$) then
        $chart_{[i,i+l+1]} \leftarrow chart_{[i,i+l+1]} \cup \{\alpha\}$;

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

	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: $\alpha \to \beta_1 \beta_2$ or $\alpha \to \gamma$ ($\beta_i \in C$, $\gamma \in \Sigma$);

- 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: $[1, 2, \text{VP} \to \text{V} \bullet \text{NP}]$;

- parser can fill in chart cells in *any* order and guarantee completeness.