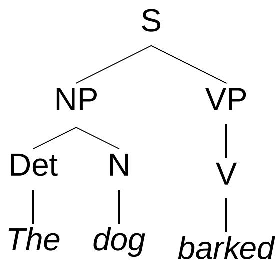


$$\begin{bmatrix} \texttt{LTOP} & h_1 \\ \texttt{INDEX} & e_2 \\ \texttt{RELS} & \left\langle \begin{bmatrix} \multicolumn{2}{l}{\textit{prpstn\_m\_rel}} \\ \texttt{LBL} & h_1 \\ \texttt{MARG} & h_3 \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{\textit{def\_q\_rel}} \\ \texttt{LBL} & h_4 \\ \texttt{ARG0} & x_5 \\ \texttt{RSTR} & h_6 \\ \texttt{BODY} & h_7 \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{\textit{“dog\_n\_rel”}} \\ \texttt{LBL} & h_8 \\ \texttt{ARG0} & x_5 \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{\textit{“bark\_v\_rel”}} \\ \texttt{LBL} & h_9 \\ \texttt{ARG0} & e_2 \\ \texttt{ARG1} & x_5 \end{bmatrix} \right\rangle \\ \texttt{HCONS} & \langle h_3 =_q h_9,\ h_6 =_q h_8 \rangle \end{bmatrix}$$

Computational Linguistics (INF2820 — Chart Parsing)

S ⟶ *NP VP*; *S* ⟶ *S PP*; *S* ⟶ *VP*

Stephan Oepen

Universitetet i Oslo

`oe@ifi.uio.no`

Top-Down vs. Bottom-Up Parsing

Top-Down (Goal-Oriented)

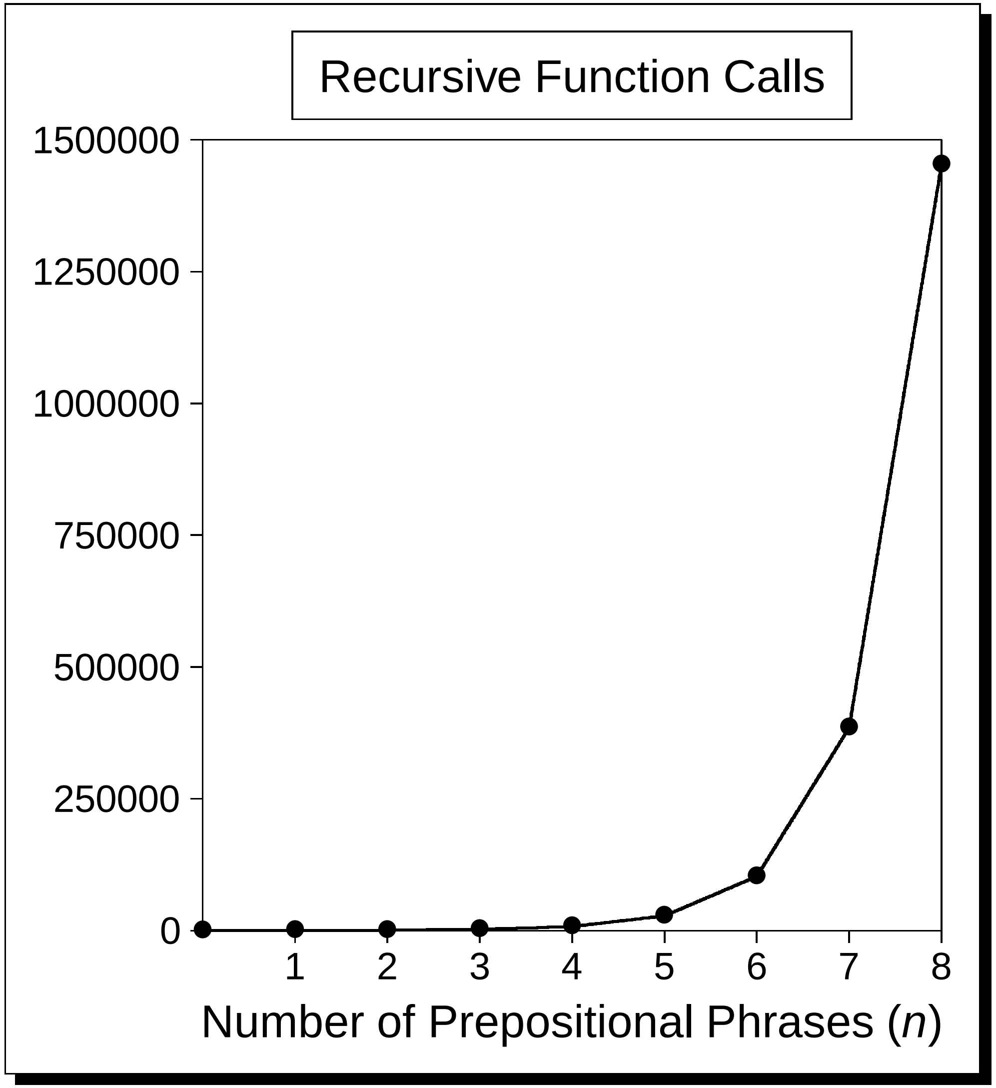
- Left recursion (e.g. a rule like ‘VP $\rightarrow$ VP PP’) causes infinite recursion;
- grammar conversion techniques (eliminating left recursion) exist, but will typically be undesirable for natural language processing applications;

$\rightarrow$ assume bottom-up as basic search strategy for remainder of the course.

Bottom-Up (Data-Oriented)

- unary (left-recursive) rules (e.g. ‘NP $\rightarrow$ 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.

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
⋮	⋮	⋮

Bounding Ambiguity — The Parse Chart

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

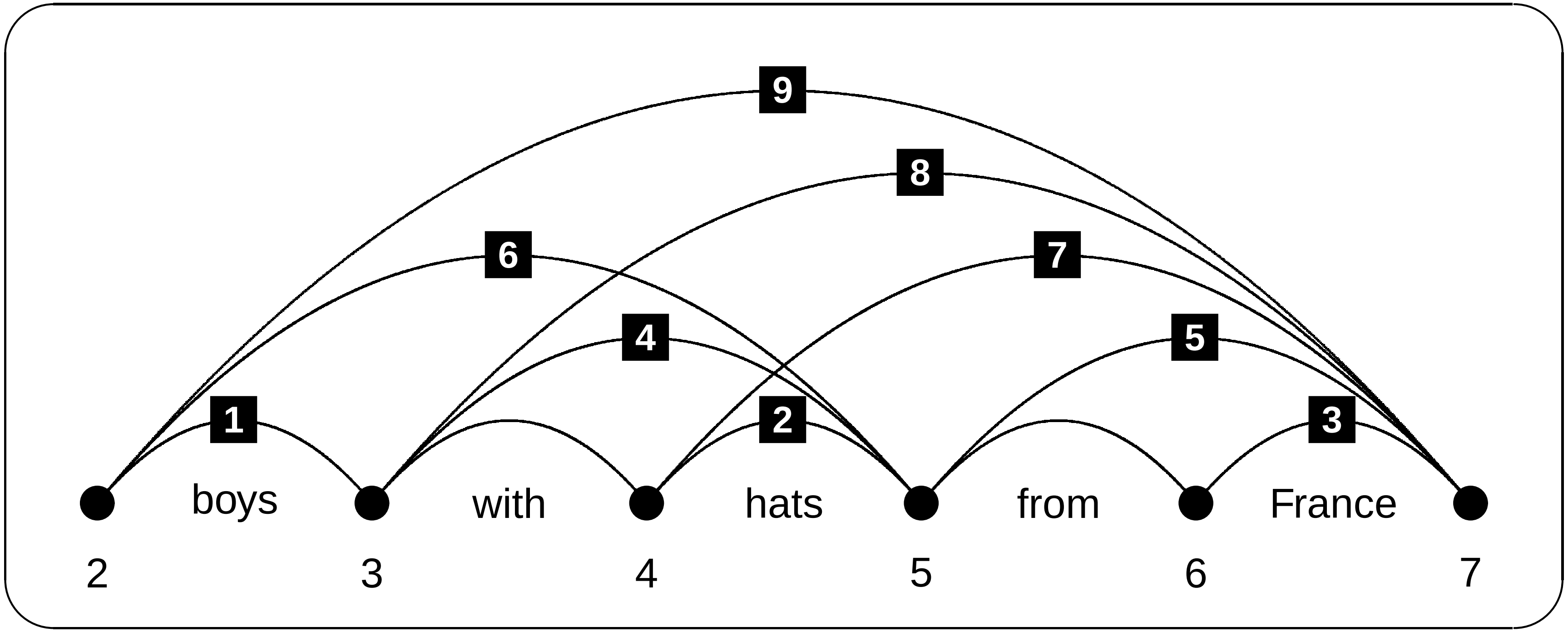


Chart Parsing — Specialized Dynamic Programming

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.

The CKY (Cocke, Kasami, & Younger) Algorithm

for ($0 \leq i < |\textit{input}|$) do
  $\textit{chart}_{[i,i+1]} \leftarrow \{\alpha \,|\, \alpha \rightarrow \textit{input}_i \in P\}$;
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
        $\textit{chart}_{[i,i+l+1]} \leftarrow \textit{chart}_{[i,i+l+1]} \cup \{\alpha\}$;

Kim adored snow in Oslo

$[0,2] \leftarrow [0,1] + [1,2]$

$\ldots$

$[0,5] \leftarrow [0,1] + [1,5]$

$[0,5] \leftarrow [0,2] + [2,5]$

$[0,5] \leftarrow [0,3] + [3,5]$

$[0,5] \leftarrow [0,4] + [4,5]$

	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.

Generalized Chart Parsing

- The parse *chart* is a two-dimensional matrix of *edges* (aka chart items);
- an edge is a (possibly partial) rule instantiation over a substring of input;
- the chart indexes edges by start and end string position (aka vertices);
- dot in rule RHS indicates degree of completion: $\alpha \rightarrow \beta_1 ... \beta_{i-1} \bullet \beta_i ... \beta_n$
- *active* edges (aka *incomplete* items) — partial RHS: $[1, 2, \text{VP} \rightarrow \text{V} \bullet \text{NP}]$;
- *passive* edges (aka *complete* items) — full RHS: $[1, 3, \text{VP} \rightarrow \text{V NP} \bullet]$;

The Fundamental Rule

$$[i,\, j,\, \alpha \rightarrow \beta_1 ... \beta_{i-1} \bullet \beta_i ... \beta_n] + [j,\, k,\, \beta_i \rightarrow \gamma^+ \bullet]$$
$$\mapsto [i,\, k,\, \alpha \rightarrow \beta_1 ... \beta_i \bullet \beta_{i+1} ... \beta_n]$$

An Example of a (Near-)Complete Chart

	1	2	3	4	5
0	NP $\rightarrow$ NP $\bullet$ PP S $\rightarrow$ NP $\bullet$ VP NP $\rightarrow$ kim $\bullet$				S $\rightarrow$ NP VP $\bullet$
1		VP $\rightarrow$ V $\bullet$ NP V $\rightarrow$ adored $\bullet$	VP $\rightarrow$ VP $\bullet$ PP VP $\rightarrow$ V NP $\bullet$		VP $\rightarrow$ VP $\bullet$ PP VP $\rightarrow$ VP PP $\bullet$ VP $\rightarrow$ V NP $\bullet$
2			NP $\rightarrow$ NP $\bullet$ PP NP $\rightarrow$ snow $\bullet$		NP $\rightarrow$ NP $\bullet$ PP NP $\rightarrow$ NP PP $\bullet$
3				PP $\rightarrow$ P $\bullet$ NP P $\rightarrow$ in $\bullet$	PP $\rightarrow$ P NP $\bullet$
4					NP $\rightarrow$ NP $\bullet$ PP NP $\rightarrow$ oslo $\bullet$

$_0$ *Kim* $_1$ *adored* $_2$ *snow* $_3$ *in* $_4$ *Oslo* $_5$

(Even) More Active Edges

	0	1	2	3
0	$S \rightarrow \bullet NP\ VP$ $NP \rightarrow \bullet NP\ PP$ $NP \rightarrow \bullet kim$	$S \rightarrow NP \bullet VP$ $NP \rightarrow NP \bullet PP$ $NP \rightarrow kim \bullet$		$S \rightarrow NP\ VP \bullet$
1		$VP \rightarrow \bullet VP\ PP$ $VP \rightarrow \bullet V\ NP$ $V \rightarrow \bullet adored$	$VP \rightarrow V \bullet NP$ $V \rightarrow adored \bullet$	$VP \rightarrow VP \bullet PP$ $VP \rightarrow V\ NP \bullet$
2			$NP \rightarrow \bullet NP\ PP$ $NP \rightarrow \bullet snow$	$NP \rightarrow NP \bullet PP$ $NP \rightarrow snow \bullet$
3				

- Include all grammar rules as *epsilon* edges in each *chart*$_{[i,i]}$ cell.
- after initialization, apply *fundamental rule* until fixpoint is reached.

Our ToDo List: Keeping Track of Remaining Work

The Abstract Goal

- Any chart parsing algorithm needs to check all pairs of adjacent edges.

A Naïve Strategy

- Keep iterating through the complete chart, combining all possible pairs, until no additional edges can be derived (i.e. the fixpoint is reached);
- frequent attempts to combine pairs multiple times: deriving 'duplicates'.

An Agenda-Driven Strategy

- Combine each pair exactly once, viz. when both elements are available;
- maintain *agenda* of new edges, yet to be checked against chart edges;
- new edges go into agenda first, add to chart upon retrieval from agenda.

Backpointers: Recording the Derivation History

	0	1	1	3
0	2: S $\rightarrow$ $\bullet$ NP VP 1: NP $\rightarrow$ $\bullet$ NP PP 0: NP $\rightarrow$ $\bullet$ kim	10: S $\rightarrow$ 8 $\bullet$ VP 9: NP $\rightarrow$ 8 $\bullet$ PP 8: NP $\rightarrow$ kim $\bullet$		17: S $\rightarrow$ 8 15 $\bullet$
1		5: VP $\rightarrow$ $\bullet$ VP PP 4: VP $\rightarrow$ $\bullet$ V NP 3: V $\rightarrow$ $\bullet$ adored	12: VP $\rightarrow$ 11 $\bullet$ NP 11: V $\rightarrow$ adored $\bullet$	16: VP $\rightarrow$ 15 $\bullet$ PP 15: VP $\rightarrow$ 11 13 $\bullet$
2			7: NP $\rightarrow$ $\bullet$ NP PP 6: NP $\rightarrow$ $\bullet$ snow	14: NP $\rightarrow$ 13 $\bullet$ PP 13: NP $\rightarrow$ snow $\bullet$
3				

- Use edges to record derivation trees: backpointers to daughters;
- a single edge can represent multiple derivations: backpointer sets.