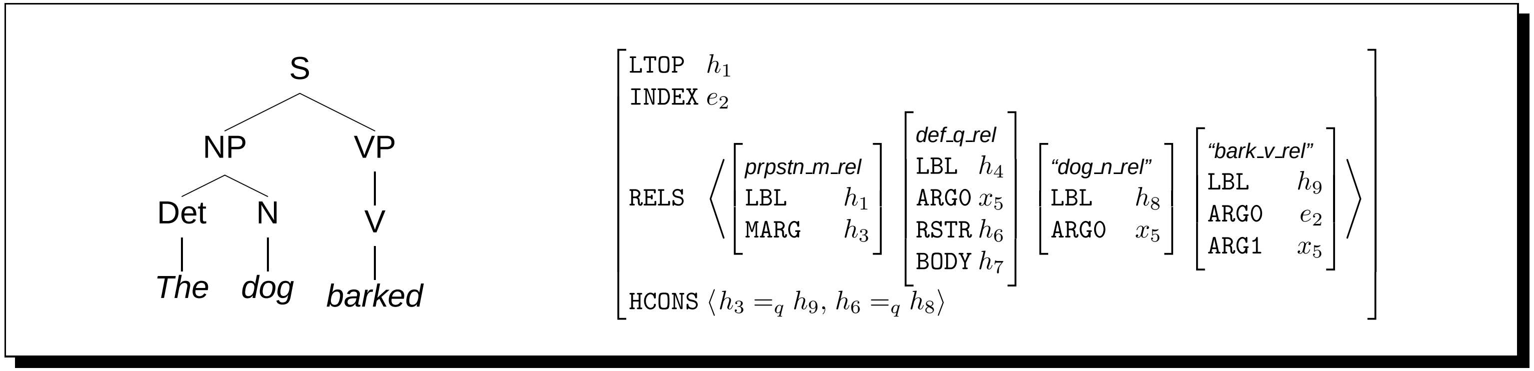


Natural Language Processing

— Context-Free Grammars and Chart Parsing —

Stephan Oepen

Universitetet i Oslo & CSLI Stanford

`oe@ifi.uio.no`

Reminding Ourselves — 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. $\{$Juan, nieve, amó, en$\}$;
- P is a set of category rewrite rules (aka *productions*), e.g.

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
NP $\rightarrow$ Juan
NP $\rightarrow$ nieve
V $\rightarrow$ amó

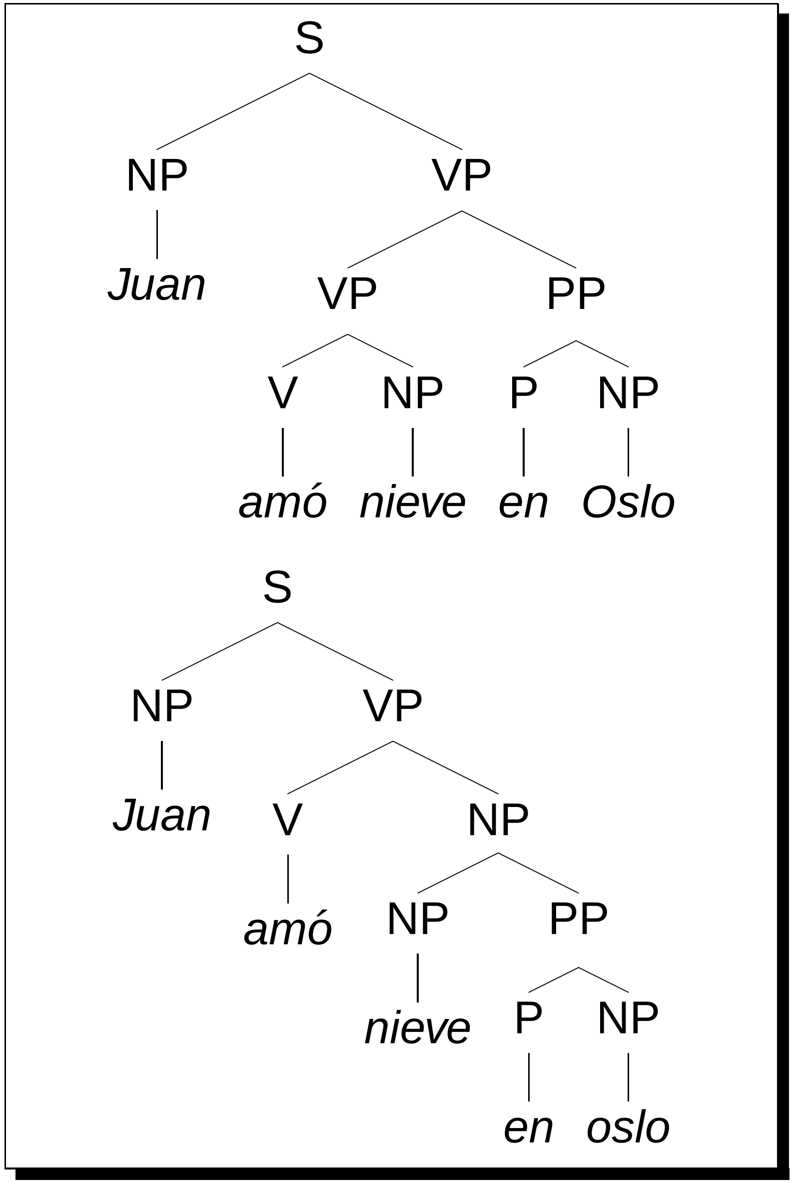
- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Parsing: Recognizing the Language of a Grammar

S → NP VP
VP → V NP
VP → VP PP
NP → NP PP
PP → P NP
NP → Juan | nieve | Oslo
V → amó
P → en

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.



Top-Down vs. Bottom-Up Parsing

Top-Down (Goal-Oriented)

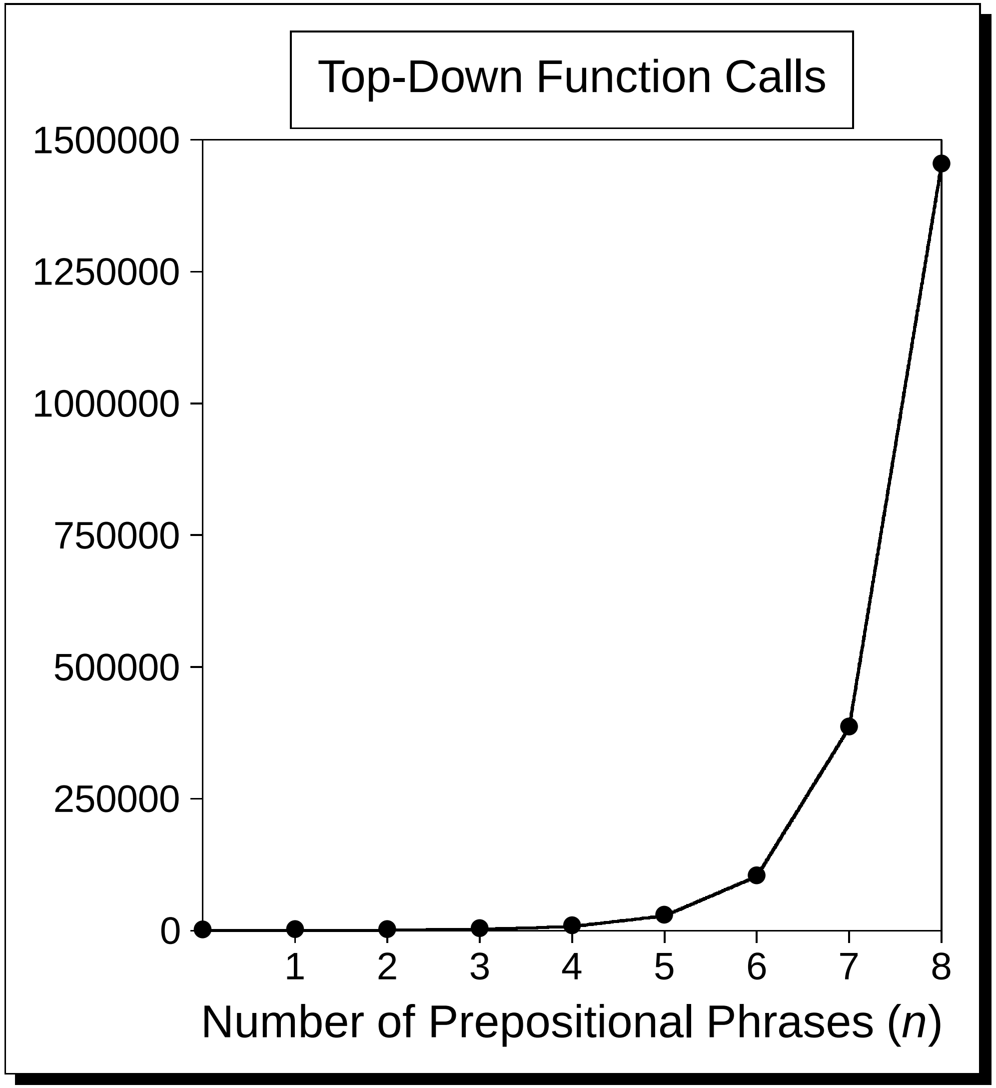
- Left recursion (e.g. the ‘VP → VP PP’ rule) causes infinite recursion;
- grammar conversion techniques (eliminating left recursion) exist, but will often be undesirable for natural language processing applications;

→ assume bottom-up as basic search strategy for NLP applications.

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.

Quantifying the Complexity of the Parsing Task

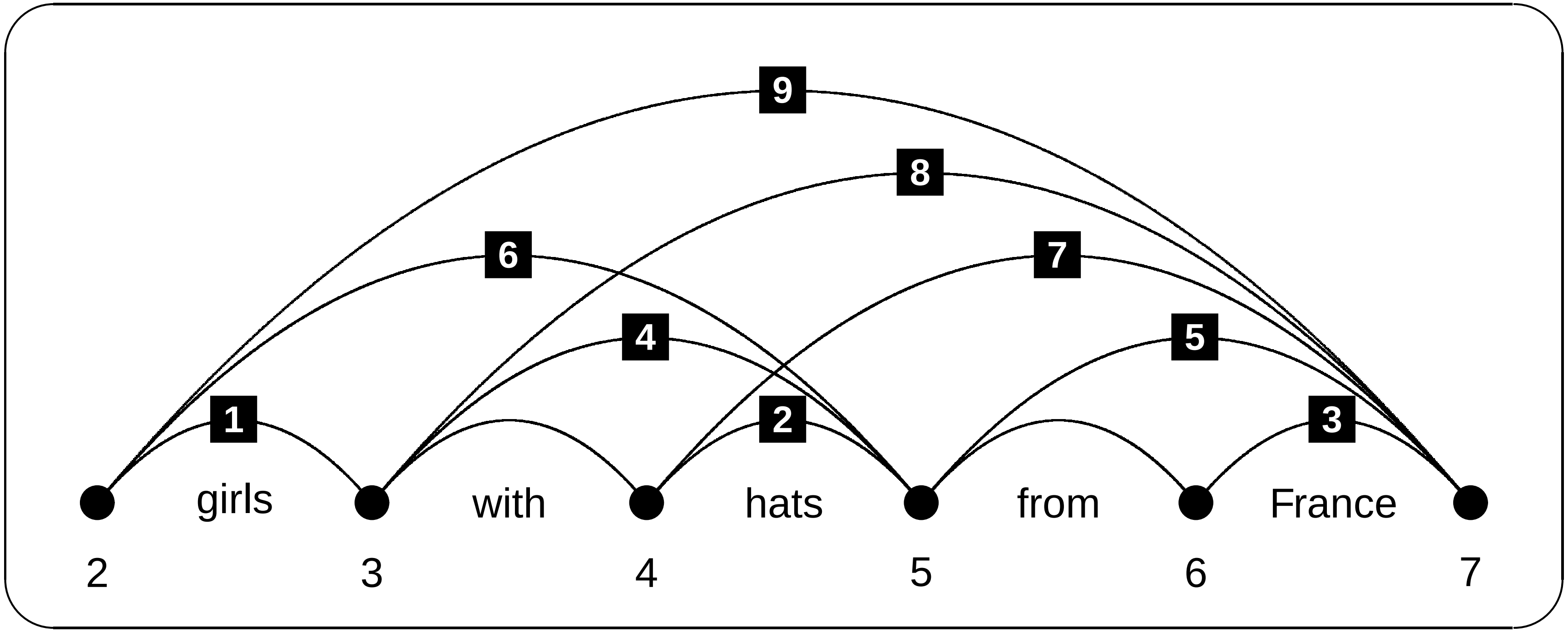


Kim adores snow (*in Oslo*)^n

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
⋮	⋮	⋮

Using the Chart to Bound Ambiguity

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



Dynamic Programming: Chart Parsing

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 same 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 ($0 \leq i < |$*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 ($\alpha \rightarrow \beta_1\, \beta_2 \in P \wedge \beta_1 \in$ *chart*$_{[i,i+j]} \wedge \beta_2 \in$ *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: $\alpha \rightarrow \beta_1\beta_2$ 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 *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

$$[x,\ y,\ \alpha \rightarrow \beta_1 ... \beta_{i-1} \bullet \beta_i ... \beta_n] + [y,\ z,\ \beta_i \rightarrow \gamma^+ \bullet]$$

$$\mapsto [x,\ z,\ \alpha \rightarrow \beta_1 ... \beta_i \bullet \beta_{i+1} ... \beta_n]$$

An Example of a (Near-)Complete Chart

	1	2	3	4	5
0	NP → NP • PP S → NP • VP NP → kim •				S → NP VP •
1		VP → V • NP V → adores •	VP → VP • PP VP → V NP •		VP → VP • PP VP → VP PP • VP → V PP •
2			NP → NP • PP NP → snow •		NP → NP • PP NP → NP PP •
3				PP → P • NP P → in •	PP → P NP •
4					NP → NP • PP NP → oslo •

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

(Even) More Active Edges

	0	1	2	3
0	S → • NP VP NP → • NP PP NP → • kim	S → NP • VP NP → NP • PP NP → kim •		S → NP VP •
1		VP → • VP PP VP → • V NP V → • adores	VP → V • NP V → adores •	VP → VP • PP VP → V NP •
2			NP → • NP PP NP → • snow	NP → NP • PP NP → snow •
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: Keeping Track of the Derivation History

	0	1	1	3
0	2: S → • NP VP 1: NP → • NP PP 0: NP → • kim	10: S → 8 • VP 9: NP → 8 • PP 8: NP → kim •		17: S → 8 15 •
1		5: VP → • VP PP 4: VP → • V NP 3: V → • adores	12: VP → 11 • NP 11: V → adores •	16: VP → 15 • PP 15: VP → 11 13 •
2			7: NP → • NP PP 6: NP → • snow	14: NP → 13 • PP 13: NP → snow •
3				

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

Ambiguity Packing in the Chart

General Idea

- Maintain only one edge for each α from i to j (the 'representative');
- record alternate sequences of daughters for α in the representative.

Implementation

- Group passive edges into *equivalence classes* by identity of α, i, and j;
- search chart for existing equivalent edge (h, say) for each new edge e;
- when h (the 'host' edge) exists, *pack* e into h to record equivalence;
- e *not* added to the chart, no derivations with or further processing of e;

$\rightarrow$ *unpacking* multiply out all alternative daughters for all result edges.