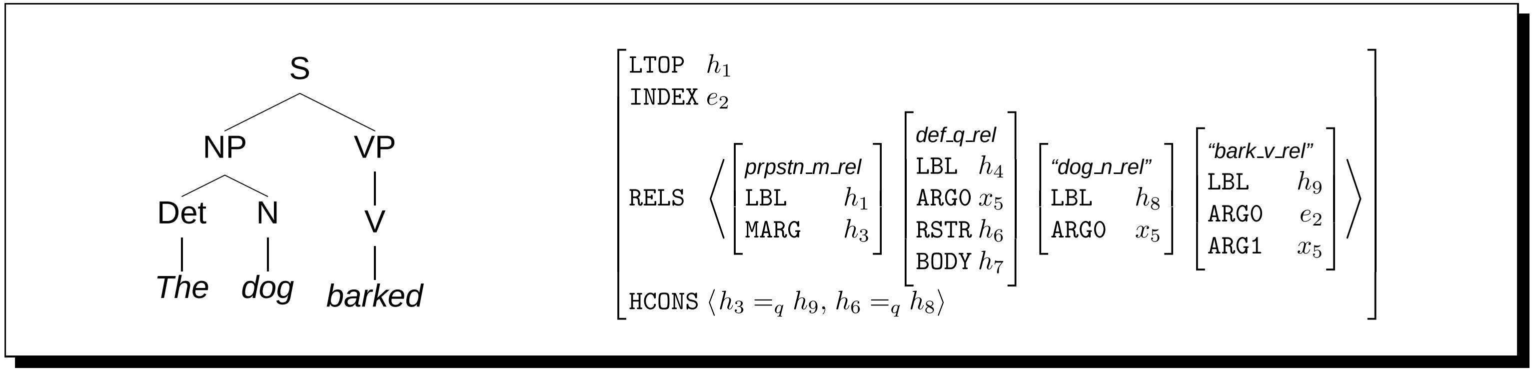


Algorithms for AI and NLP (INF4820 — Parsing)

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

Stephan Oepen and Jan Tore Lønning

Universitetet i Oslo

{ oe | jtl }@ifi.uio.no

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

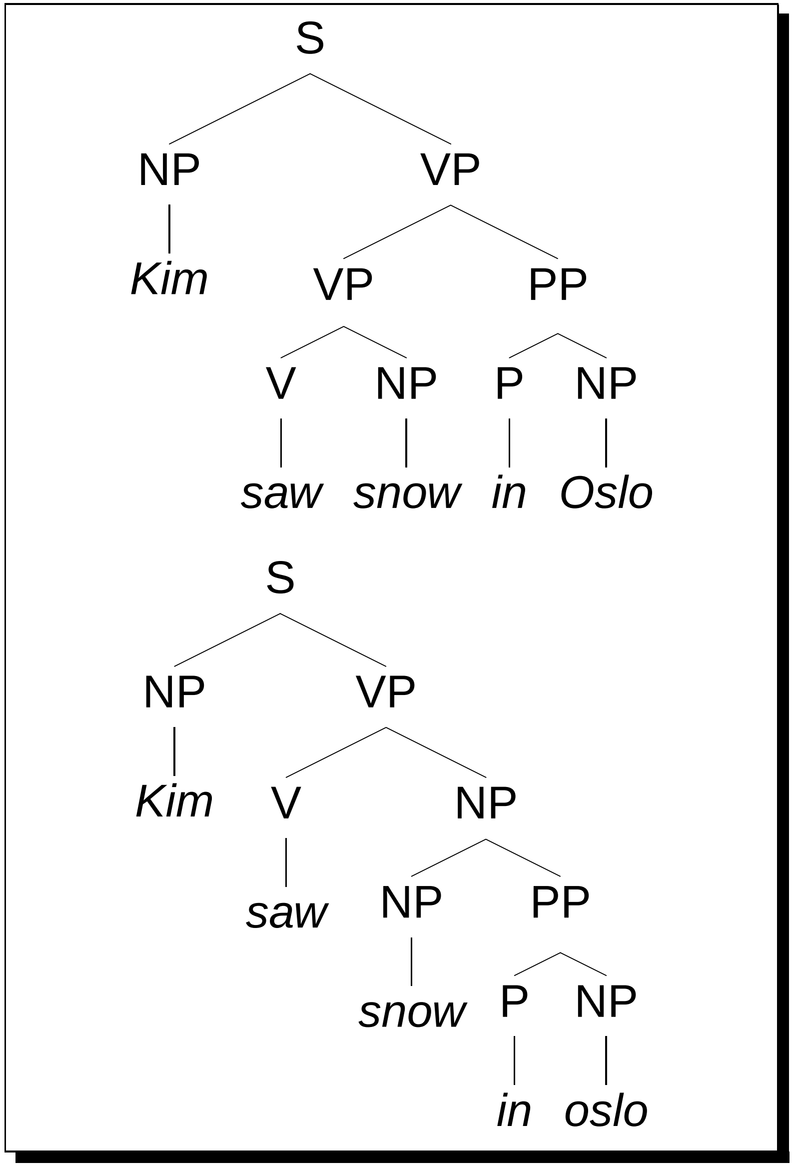
- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Parsing: Recognizing the Language of a Grammar

S → NP VP
VP → V | V NP | VP PP
NP → NP PP
PP → P NP
NP → Kim | snow | Oslo
V → saw
P → 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.



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.

A Recursive Descent Parser

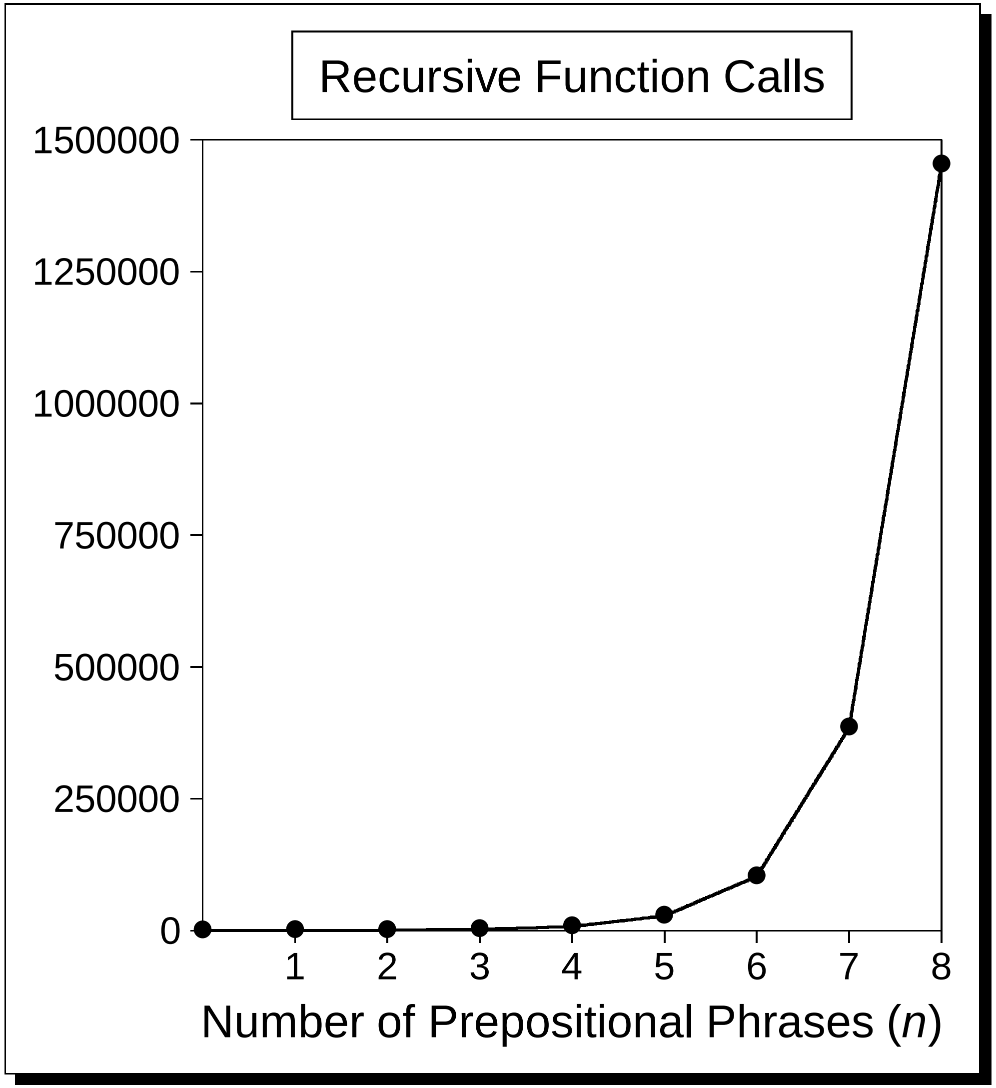
```
(defun parse (input goal)
  (if (equal (first input) goal)
    (list (make-state :tree (first input) :input (rest input)))
    (loop
        for rule in (rules-rewriting goal)
        append (instantiate (rule-lhs rule) nil (rule-rhs rule) input))))
```

```
(defun instantiate (lhs analyzed unanalyzed input)
  (if (null unanalyzed)
    (list (make-state :tree (make-tree :root lhs :daughters analyzed)
                      :input input))
    (loop
        for parse in (parse input (first unanalyzed))
        append (instantiate
                lhs
                (append analyzed (list (state-tree parse)))
                (rest unanalyzed)
                (state-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
      ...
```

Quantifying the Complexity of the Parsing Task



Kim adores snow (in Oslo)^n

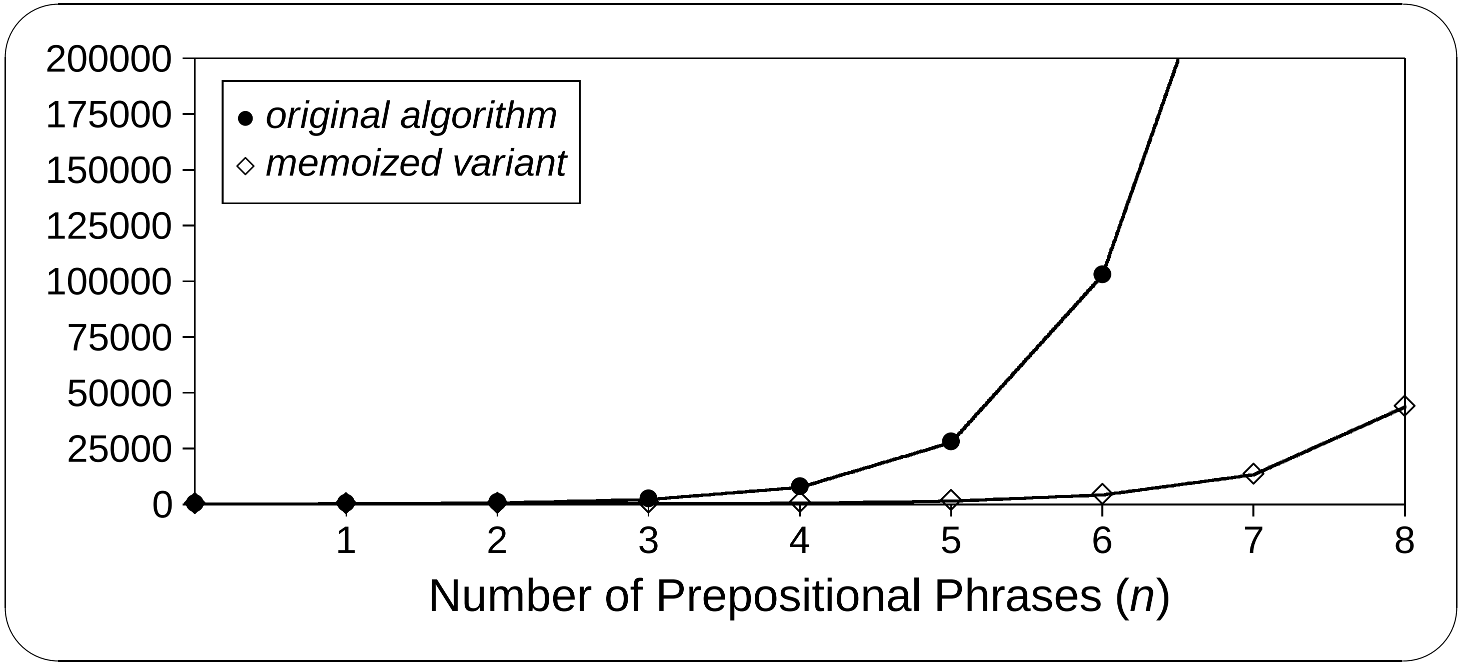
n	**trees**	**calls**
0	1	46
1	2	170
2	5	593
3	14	2,093
4	42	7,539
5	132	27,627
6	429	102,570
7	1430	384,566
8	4862	1,452,776
⋮	⋮	⋮

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;

→ 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 $\rightarrow$ 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.

A Bottom-Up Variant (1 of 2)

- Work upwards from string; successively combine words or phrases into larger phrases;
- use all grammar rules that have the (currently) next input word as β_1 in their RHS;
- recursively attempt to instantiate the remaining part of each rule RHS (β_i; $2 \leq i \leq n$);
- when a rule $\alpha \rightarrow \beta_i^+$ has been completely instantiated, attempt all rules starting in α;
- for each (remaining) input (suffix), derive all trees that span a prefix or all of the input.

```
(defun parse (input)
  (when input
    (loop
        for rule in (rules-starting-in (first input))
        append (instantiate (rule-lhs rule)
                            (list (first (rule-rhs rule)))
                            (rest (rule-rhs rule))
                            (rest input)))))
```

A Bottom-Up Variant (2 of 2)

```
(defun instantiate (lhs analyzed unanalyzed input)
  (if (null unanalyzed)
    (let ((tree (make-tree :root lhs :daughters analyzed)))
      (cons (make-state :tree tree :input input)
            (loop
                for rule in (rules-starting-in lhs)
                append
                  (instantiate (rule-lhs rule)
                               (list tree)
                               (rest (rule-rhs rule))
                               input))))
    (loop
        for state in (parse input)
        when (equal (tree-root (state-tree state))
                    (first unanalyzed))
        append (instantiate lhs
                            (append analyzed (list (state-tree state)))
                            (rest unanalyzed)
                            (state-input state)))))
```

Chart Parsing — Specialized Dynamic Programming

Basic Notions

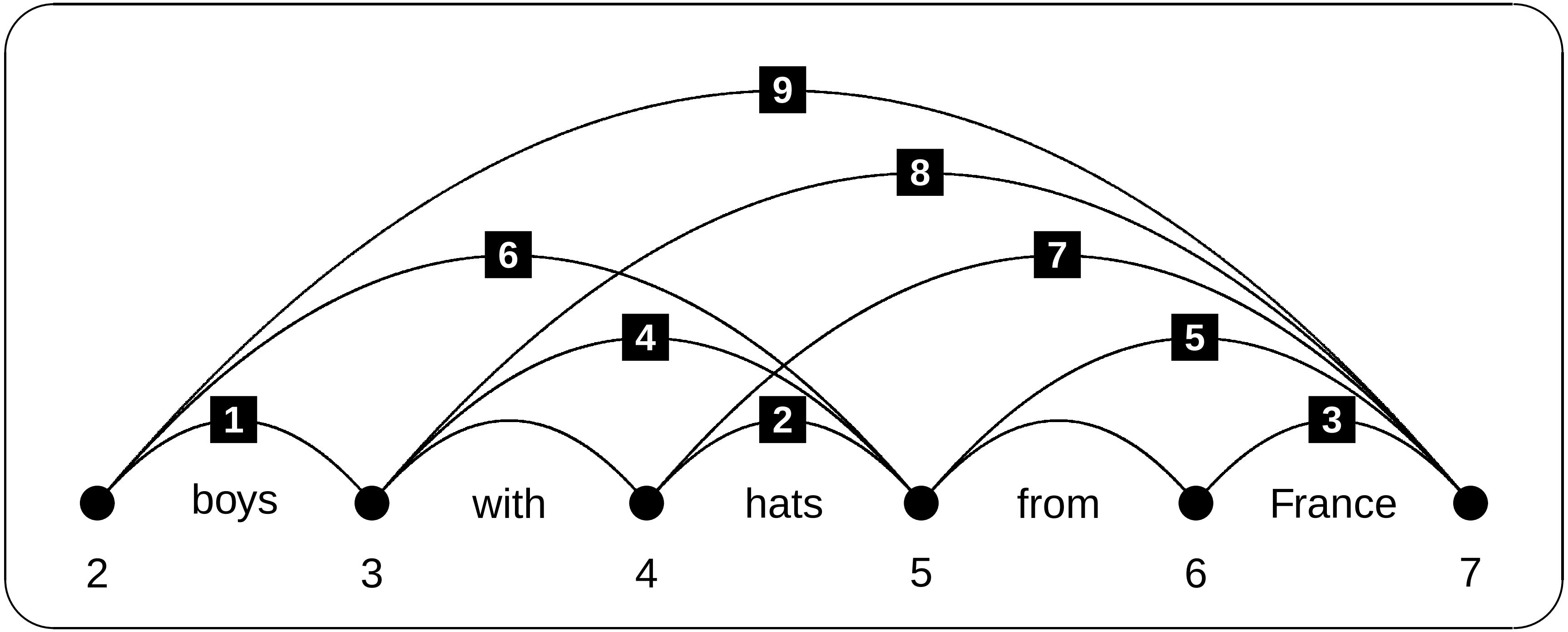
- Use *chart* to record partial analyses, indexing them by string positions;
- count inter-word vertices; CKY: chart row is *start*, column *end* vertex;
- treat multiple ways of deriving the same category for some substring as *equivalent*; pursue only once when combining with other constituents.

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
  $\textit{chart}_{[i,i+1]} \leftarrow \{\alpha \,|\, \alpha \rightarrow \textit{input}_i \in P\}$;
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.

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 → NP • PP S → NP • VP NP → kim •				S → NP VP •
1		VP → V • NP V → adored •	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* $_1$ *adored* $_2$ *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 → • adored	VP → V • NP V → adored •	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: Recording the Derivation History

	0	1	2	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 → • adored	12: VP → 11 • NP 11: V → adored •	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.

Chart Elements: The Edge Structure

`#[`*id* `:` `(`*i*–*j*`)` α `-->` *edge*$_1$ ... *edge*$_i$ `.` β_{i+1} ... β_n $\{$ *alternate*$_1$ ... *alternate*$_n$ $\}^*$ `]`

Components of the *edge* Structure

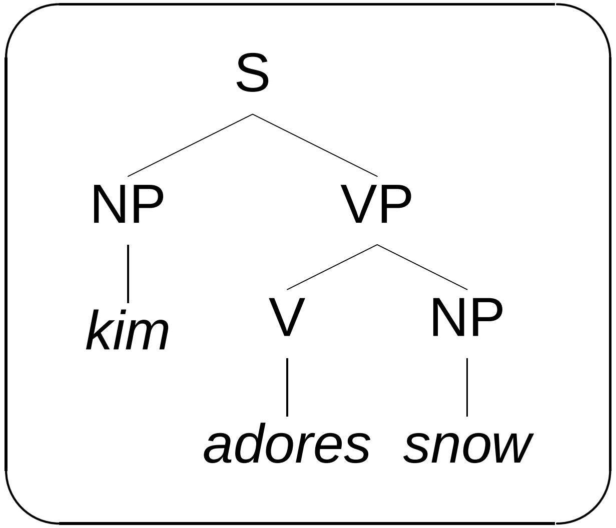
- *id* unique edge identifier (automatically assigned my `make-edge()`);
- *i* and *j* starting and ending string index (chart vertices) for this edge;
- α category of this edge (from the set C of non-terminal symbols);
- *edge*$_1$ ... *edge*$_i$ (list of) daughter edges (for β_1 ... β_i) instantiated so far;
- β_{i+1} ... β_n (list of) remaining categories in rule RHS to be instantiated;
- *alternate*$_1$ ... *alternate*$_n$ alternative derivation(s) for α from i to j;

$\rightarrow$ implemented using `defstruct()` (plus suitable pretty printing routine).

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.

Ambiguity Resolution Remains a (Major) Challenge

The Problem

- With broad-coverage grammars, even moderately complex sentences typically have multiple analyses (tens or hundreds, rarely thousands);
- unlike in grammar writing, exhaustive parsing is useless for applications;
- identifying the 'right' (intended) analysis is an 'AI-complete' problem;
- inclusion of (non-grammatical) sortal constraints is generally undesirable.

Typical Approaches

- Design and use statistical models to select among competing analyses;
- for string S, some analyses T_i are more or less likely: maximize $P(T_i|S)$;

$\rightarrow$ Probabilistic Context Free Grammar (PCFG) is a CFG plus probabilities.

Probability Theory and Linguistics?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Probability Theory and Linguistics?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

Probability Theory and Linguistics?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

But it must be recognized that the notion 'probability' of a sentence is an entirely useless one, under any known interpretation of this term. (Noam Chomsky, 1969)

Probability Theory and Linguistics?

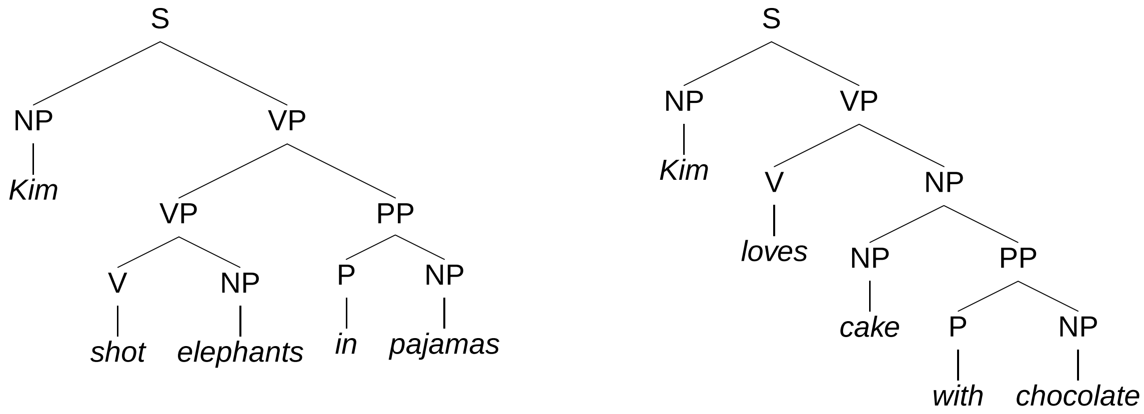
The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

But it must be recognized that the notion 'probability' of a sentence is an entirely useless one, under any known interpretation of this term. (Noam Chomsky, 1969)

Every time I fire a linguist, system performance improves. (Fredrick Jelinek, 1980s)

A (Simplified) PCFG Estimation Example



P(RHS|LHS)	**CFG Rule**
	S → NP VP
	VP → VP PP
	VP → V NP
	PP → P NP
	NP → NP PP
	VP → V

- Estimate rule probability from observed distribution;

→ conditional probabilities:

$$P(\mathrm{RHS}|\mathrm{LHS}) = \frac{C(\mathrm{LHS}, \mathrm{RHS})}{C(\mathrm{LHS})}$$