INF4820: Algorithms for Artificial Intelligence and Natural Language Processing

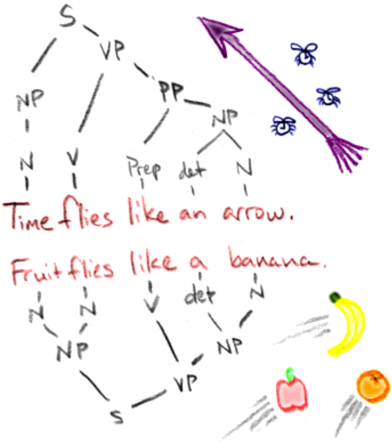
Chart Parsing

Stephan Oepen & Milen Kouylekov

Language Technology Group (LTG)

October 30, 2014

Syntactic Structure and Ambiguity



(Courtesy of the *Speculative Grammarian*, *–the journal of satirical linguistics*.)

Context-Free Grammars

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - $\{\text{S}, \text{NP}, \text{VP}, \text{V}\}$
- Σ is the vocabulary (aka *terminals*),
  - $\{\text{Kim}, \text{snow}, \text{adores}, \text{in}\}$
- P is a set of category rewrite rules (aka *productions*)

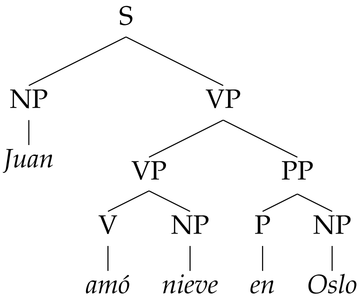
$$\begin{array}{lr} \text{S} \rightarrow \text{NP VP} & \text{NP} \rightarrow \text{Kim} \\ \text{VP} \rightarrow \text{V NP} & \text{NP} \rightarrow \text{snow} \\ & \text{V} \rightarrow \text{adores} \end{array}$$

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

Justice for Dr. Kouylekov

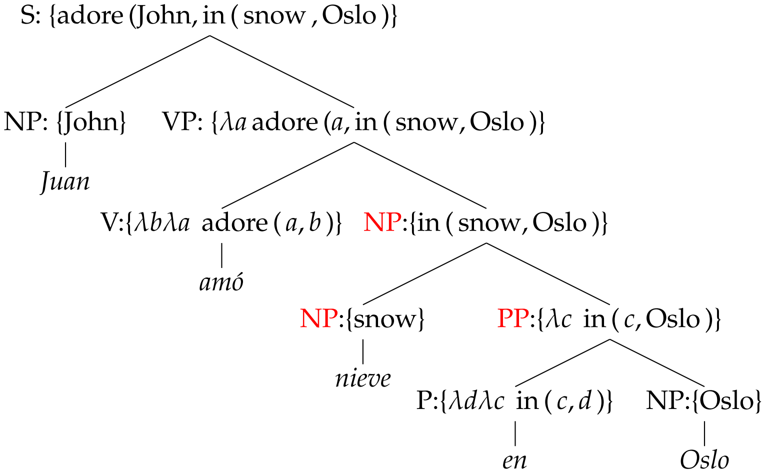
The Grammar of Spanish

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



Juan amó nieve en Oslo

Another Interpretation



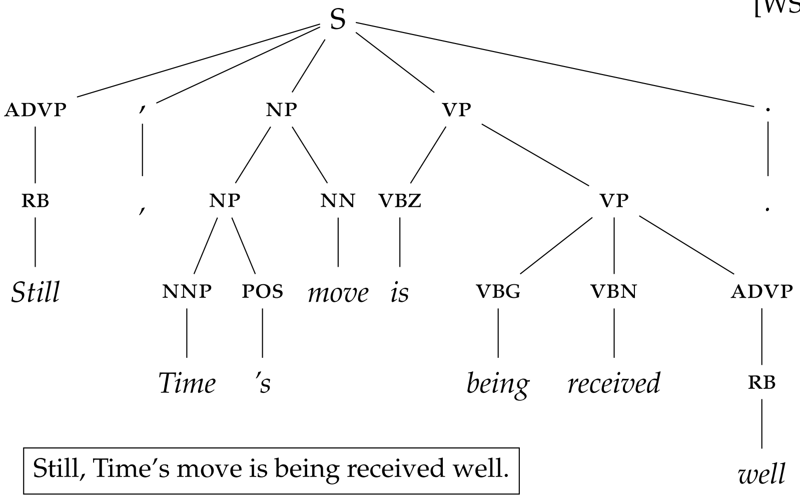
$$\text{NP} \rightarrow \text{NP PP} \; \{ \, \text{PP} \, (\, \text{NP} \,) \, \}$$

Probabilitic Context-Free Grammars

- We are interested, not just in which trees apply to a sentence, but also to which tree is **most likely**.
- Probabilistic context-free grammars (PCFGs) augment CFGs by adding probabilities to each production, e.g.
  - S → NP VP 0.6
  - S → NP VP PP 0.4
- These are conditional probabilities — the probability of the right hand side (RHS) given the left hand side (LHS)
  - $P(S \rightarrow NP\ VP) = P(NP\ VP|S)$
- We can learn these probabilities from a treebank, again using Maximum Likelihood Estimation.

Estimating PCFGs (1/3)

[WSJ 2350]



Still, Time’s move is being received well.

Estimating PCFGs (2/3)

```
(S
   (ADVP (RB "Still"))
   (|,| ",")
   (NP
      (NP (NNP "Time") (POS "’s"))
      (NN "move"))
    (VP
      (VBZ "is")
      (VP
         (VBG "being")
         (VP
           (VBN "received")
           (ADVP (RB "well")))))
   (\. "."))
```

RB → Still	1
AVP → RB	2
|,| → ,	1
NNP → Time	1
POS → ’s	1
NP → NNP POS	1
NN → move	1
NP → NP NN	1
VBZ → is	1
VBG → being	1
VBN → received	1
RB → well	1
VP → VBN ADVP	1
VP → VBG VP	1
\\. → .	1
S → ADVP |,| NP VP \\.	1
START → S	1

Estimating PCFGs (3/3)

Once we have counts of all the rules, we turn them into probabilities.

S → ADVP |,| NP VP \\.	50	S → NP VP \\.	400
S → NP VP PP \\.	350	S → VP !	100
S → NP VP S \\.	200	S → NP VP	50

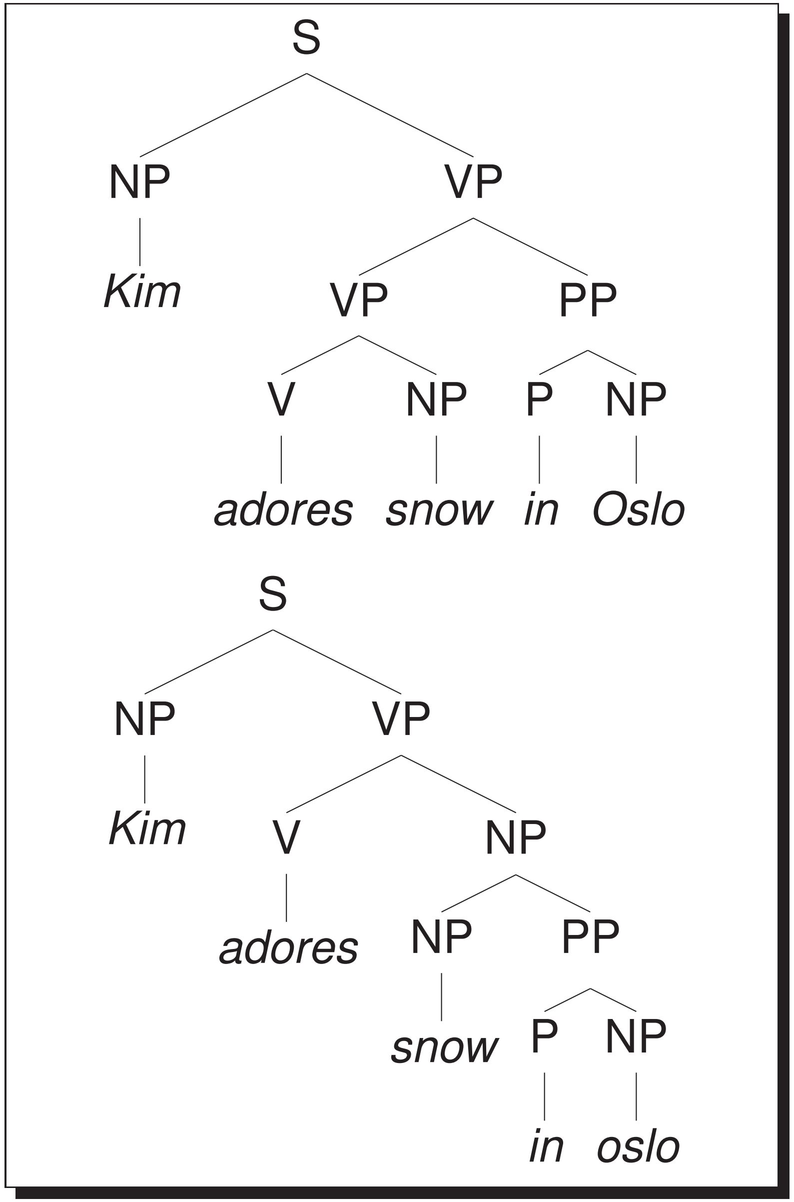
$$
\begin{aligned}
P(S \rightarrow ADVP\ |,|\ NP\ VP\ \backslash.) &\approx \frac{C(S \rightarrow ADVP\ |,|\ NP\ VP\ \backslash.)}{C(S)} \\
&= \frac{50}{1150} \\
&= 0.0435
\end{aligned}
$$

Parsing with CFGs: Moving to a Procedural View

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



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

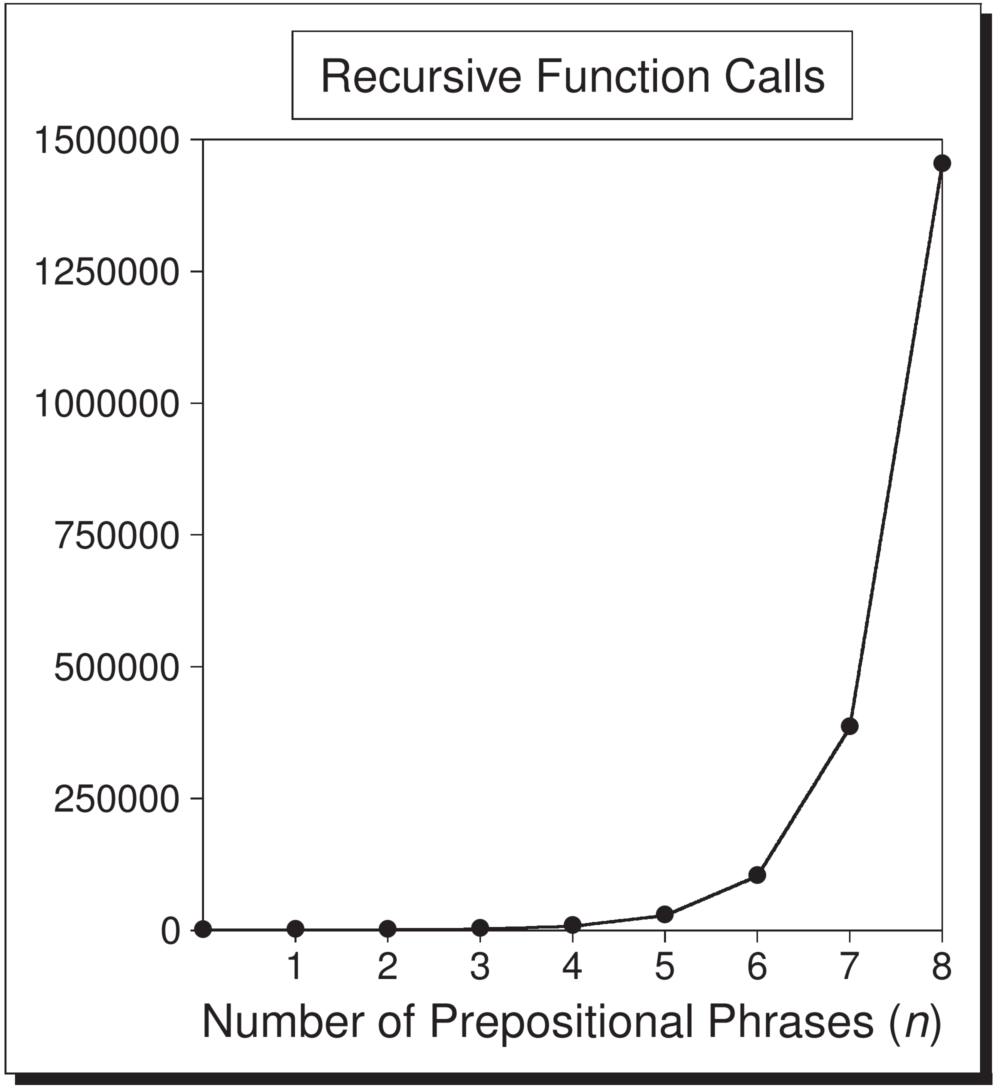
- Each result records a (partial) tree and remaining input to be parsed;
- complete results consume the full input string and are rooted in S;
- whenever a RHS is fully instantiated, a new tree is built and returned;
- all results at each level are combined and successively accumulated.

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 ((tree (cons goal analyzed)))
      (list (make-parse :tree tree :input input)))
    (loop
        for parse in (parse input (first unanalyzed))
        append (extend-parse
                goal (append analyzed (list (parse-tree 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 $\rightarrow$ VP PP') causes infinite recursion;
- search is uninformed by the (observable) input: can hypothesize many unmotivated sub-trees, assuming terminals (words) that are not present;

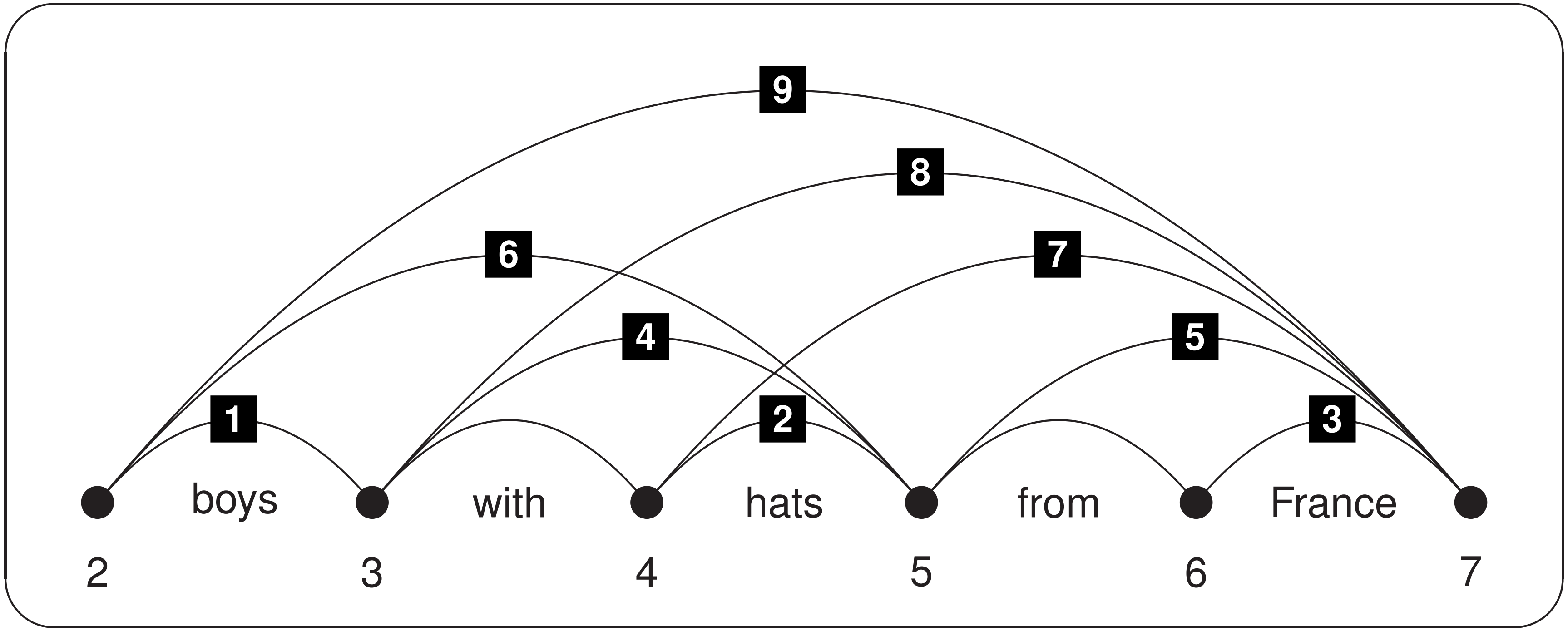
$\rightarrow$ assume bottom-up as basic search strategy for remainder of the course.

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 Key Insight: Local Ambiguity

- For many substrings, more than one way 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].



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
        $\textit{chart}_{[i,i+l+1]} \leftarrow \textit{chart}_{[i,i+l+1]} \cup \{\alpha\}$;

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

	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.

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.

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: $\alpha \rightarrow \beta_1 \ldots \beta_{i-1} \bullet \beta_i \ldots \beta_n$;
- *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

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

$$\mapsto [i,\ k,\ \alpha \rightarrow \beta_1 \ldots \beta_i \bullet \beta_{i+1} \ldots \beta_n]$$

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.

Combinatorics: 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.

In Conclusion—What Happened this Week

Syntactic Structure

- Languages (formal or natural) exhibit complex, hierarchical structures;
- grammars encode rules of the language: dominance and sequencing;
- context-free grammar ‘generates’ a language: strings and derivations;
- ambiguity in natural language grows exponentially: a search problem;
- bounding (or ‘packing’) of local ambiguity madantory for tractability;
- chart parsing uses dynamic programming: free order of computation.

Coming up Next

- Viterbi adaptation to parse forest; PTB parsing; parser evaluation; quiz.