Statistical NLP
Winter 2008

Lecture 11: Parsing III

Roger Levy
UC San Diego

[Dan Klein, *merci beaucoup*!]

(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don’t know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.
A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
  - Nodes: positions between words
  - Edges: spans of words with labels, represent the set of trees over those words rooted at x
  - A chart: records which edges we’ve built
  - An agenda: a holding pen for edges (a queue)
- We’re going to figure out:
  - What edges can we build?
  - All the ways we built them.

Diagram:
- Nodes: positions between words
- Edges: spans of words with labels
- Chart: records which edges we’ve built
- Agenda: a holding pen for edges (a queue)
Word Edges

- An edge found for the first time is called discovered. Edges go into the agenda on discovery.
- To initialize, we discover all word edges.

AGENDA

critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

CHART [EMPTY]

critics      write      reviews      with      computers
Unary Projection

- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it

```
<table>
<thead>
<tr>
<th></th>
<th>critics[0,1]</th>
<th>write[1,2]</th>
<th>reviews[2,3]</th>
<th>with[3,4]</th>
<th>computers[4,5]</th>
</tr>
</thead>
</table>
```

Diagram:

```
critics  write  reviews  with  computers
0        1       2        3       4
```
The “Fundamental Rule”

- When we pop edges off of the agenda:
  - Check for unary projections (NNS $\rightarrow$ critics, NP $\rightarrow$ NNS)

\[ Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j] \]

- Combine with edges already in our chart (this is sometimes called the fundamental rule)

\[ Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y Z \text{ form } X[i,k] \]

- Enqueue resulting edges (if newly discovered)
- Record backtraces (called traversals)
- Stick the popped edge in the chart

- Queries a chart must support:
  - Is edge $X:[i,j]$ in the chart?
  - What edges with label $Y$ end at position $j$?
  - What edges with label $Z$ start at position $i$?
An Example

Critics write reviews with computers.
Exploiting Substructure

- Each edge records all the ways it was built (locally)
  - Can recursively extract trees
  - A chart may represent too many parses to enumerate (how many?)
Order Independence

• **A nice property:**
  • It doesn’t matter what policy we use to order the agenda (FIFO, LIFO, random).

• **Why? Invariant: before popping an edge:**
  • Any edge $X[i,j]$ that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
  • Convince yourselves this invariant holds!

• **This will not be true weighted parsers:**
  • Instead must also insure that an edge has best score when added to the chart
  • Sufficient (but not necessary) to order agenda items by current best score
Problems with PCFGs?

If we do no annotation, these trees differ only in one rule:
- $\text{VP} \rightarrow \text{VP PP}$
- $\text{NP} \rightarrow \text{NP PP}$

Parse will go one way or the other, regardless of words.

We’ll look at two ways to address this:
- Sensitivity to specific words through *lexicalization*
- Sensitivity to structural configuration with *unlexicalized methods*
Problems with PCFGs

- What’s different between basic PCFG scores here?
- What (lexical) correlations need to be scored?
Problems with PCFGs

- Another example of PCFG indifference
  - Left structure far more common
  - How to model this?
  - Really structural: “chicken with potatoes with gravy”
  - Lexical parsers model this effect, though not by virtue of being lexical
Lexicalized Trees

- Add “headwords” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - NP:
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - VP:
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child

```
S
  NP
    DT the
    NN lawyer
  VP
    Vt questioned
    NP
      DT the
      NN witness

S(questioned)
  NP(lawyer)
    DT the
    NN lawyer
  VP(questioned)
    Vt questioned
    NP(witness)
      DT the
      NN witness
```
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like

  \[ VP(saw) \rightarrow VBD(saw) \text{ NP-C(her) NP(today)} \]

- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps
Lexical Derivation Steps

- Simple derivation of a local tree [simplified Charniak 97]

```
VP[saw]
  VBD[saw] NP[her] NP[today] PP[on]

It's markovization again!

Still have to smooth with mono- and non-lexical backoffs
```

\[
P(\text{STOP}|\text{VBD[saw]}, \text{VP}, \text{PP})
\]

\[
P(\text{PP[on]}|\text{VBD[saw]}, \text{VP}, \text{NP})
\]

\[
P(\text{NP[today]}|\text{VBD[saw]}, \text{VP}, \text{NP})
\]

\[
P(\text{NP[her]}|\text{VBD[saw]}, \text{VP}, \text{START})
\]

\[
P(\text{VBD[saw]}|\text{VP[saw]})
\]
Lexical Derivation Steps

- Another derivation of a local tree [Collins 99]

Choose a head tag and word

Choose a complement bag

Generate children (incl. adjuncts)

Recursively derive children
Naïve Lexicalized Parsing

• Can, in principle, use CKY on lexicalized PCFGs
  • $O(Rn^3)$ time and $O(Sn^2)$ memory
  • But $R = rV^2$ and $S = sV$
  • Result is completely impractical (why?)
  • Memory: 10K rules * 50K words * (40 words)$^2$ * 8 bytes $\approx$ 6TB

• Can modify CKY to exploit lexical sparsity
  • Lexicalized symbols are a base grammar symbol and a pointer into the input sentence, not any arbitrary word
  • Result: $O(rn^5)$ time, $O(sn^3)$
  • Memory: 10K rules * (40 words)$^3$ * 8 bytes $\approx$ 5GB
Lexicalized CKY

\[
\text{bestScore}(X, i, j, h)
\]

\[
\text{if } (j = i+1) \rightarrow \text{return } \text{tagScore}(X, s[i])
\]

\[
\text{else} \rightarrow \text{return } \max \max_{k, X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h] Z[h']) \ast \\
\max_{k, X \rightarrow YZ} \text{bestScore}(Y,i,k,h) \ast \\
\text{bestScore}(Z,k,j,h')
\]

\[
\max \max_{k, X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h'] Z[h]) \ast \\
\text{bestScore}(Y,i,k,h') \ast \\
\text{bestScore}(Z,k,j,h)
\]
Quartic Parsing

- Turns out, you can do better [Eisner 99]
- Gives an O(n^4) algorithm
- Still prohibitive in practice if not pruned
Dependency Parsing

- Lexicalized parsers can be seen as producing *dependency trees*

Each local binary tree corresponds to an attachment in the dependency graph.
Dependency Parsing

• Pure dependency parsing is only cubic [Eisner 99]

• Some work on non-projective dependencies
  • Common in, e.g. Czech parsing
  • Can do with MST algorithms [McDonald and Pereira 05]
    • Leads to $O(n^3)$ or even $O(n^2)$ [McDonald et al., 2005]
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ CKY
  - Remember only a few hypotheses for each span $<i,j>$.
  - If we keep $K$ hypotheses at each span, then we do at most $O(nK^2)$ work per span (why?)
  - Keeps things more or less cubic

- Side note/hack: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)
Pruning with a PCFG

• The Charniak parser prunes using a two-pass approach [Charniak 97+
  • First, parse with the base grammar
  • For each X[:i,j] calculate $P(X|i,j,s)$
    • This isn’t trivial, and there are clever speed ups
  • Second, do the full $O(n^5)$ CKY
    • Skip any X [:i,j] which had low (say, < 0.0001) posterior
    • Avoids almost all work in the second phase!
    • Currently the fastest lexicalized parser

• Charniak et al 06: can use more passes
• Petrov et al 07: can use many more passes
Pruning with A*

- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]
Factory payrolls fell in Sept.
A* Speedup

- Total time dominated by calculation of A* tables in each projection… $O(n^3)$
Results

- Some results
  - Collins 99 – 88.6 F1 (generative lexical)
  - Charniak and Johnson 05 – 89.7 / 91.3 F1 (generative lexical / reranked)
  - Petrov et al 06 – 90.7 F1 (generative unlexical)
  - McClosky et al 06 – 92.1 F1 (gen + rerank + self-train)

- However
  - Bilexical counts rarely make a difference (why?)
  - Gildea 01 – Removing bilexical counts costs < 0.5 F1

- Bilexical vs. monolexical vs. smart smoothing
Unlexicalized methods

• So far we have looked at the use of \textit{lexicalized} methods to fix PCFG independence assumptions

• Lexicalization creates new complications of \textit{inference} (computational complexity) and \textit{estimation} (sparsity)

• There are lots of improvements to be made without resorting to lexicalization
PCFGs and Independence

- Symbols in a PCFG define independence assumptions:

\[ S \rightarrow NP \ VP \]
\[ NP \rightarrow DT \ NN \]

- At any node, the material inside that node is independent of the material outside that node, given the label of that node.
- Any information that statistically connects behavior inside and outside a node must flow through that node.
Non-Independence I

- Independence assumptions are often too strong.
  
  All NPs
  
  NPs under S
  
  NPs under VP

- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!
Non-Independence II

- Who cares?
  - NB, HMMs, all make false assumptions!
  - For generation, consequences would be obvious.
  - For parsing, does it impact accuracy?

- Symptoms of overly strong assumptions:
  - Rewrites get used where they don’t belong.
  - Rewrites get used too often or too rarely.

\[
\text{In the PTB, this construction is for possessives}
\]
• We can relax independence assumptions by encoding dependencies into the PCFG symbols:

Parent annotation

[Johnson 98]

Marking possessive NPs

• What are the most useful “features” to encode?
Annotations

- Annotations split the grammar categories into sub-categories (in the original sense).

- Conditioning on history vs. annotating
  - $P(NP^S \rightarrow PRP)$ is a lot like $P(NP \rightarrow PRP \mid S)$
    - Or equivalently, $P(PRP \mid NP, S)$
  - $P(NP-POS \rightarrow NNP POS)$ isn’t history conditioning.

- Feature / unification grammars vs. annotation
  - Can think of a symbol like $NP^{NP-POS}$ as $NP \ [parent:NP, +POS]\$

- After parsing with an annotated grammar, the annotations are then stripped for evaluation.
Lexicalization

- Lexical heads important for certain classes of ambiguities (e.g., PP attachment):

- Lexicalizing grammar creates a much larger grammar. (cf. next week)
  - Sophisticated smoothing needed
  - Smarter parsing algorithms
  - More data needed

- How necessary is lexicalization?
  - Bilexical vs. monolexical selection
  - Closed vs. open class lexicalization
Unlexicalized PCFGs

- What is meant by an “unlexicalized” PCFG?
  - Grammar not systematically specified to the level of lexical items
    - NP [stocks] is not allowed
    - NP^S-CC is fine
  - Closed vs. open class words (NP^S [the])
    - Long tradition in linguistics of using function words as features or markers for selection
    - Contrary to the bilexical idea of semantic heads
    - Open-class selection really a proxy for semantics

- It’s kind of a gradual transition from unlexicalized to lexicalized (but heavily smoothed) grammars.
Typical Experimental Setup

- **Corpus:** Penn Treebank, WSJ

<table>
<thead>
<tr>
<th>Training:</th>
<th>Development:</th>
<th>Test:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sections</td>
<td>section</td>
<td></td>
</tr>
<tr>
<td>02-21</td>
<td>22 (here, first 20 files)</td>
<td>23</td>
</tr>
</tbody>
</table>

- Accuracy – F1: harmonic mean of per-node labeled precision and recall.
- Here: also size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: NP → NP CC
Multiple Annotations

- Each annotation done in succession
  - Order does matter
  - Too much annotation and we’ll have sparsity issues
Horizontal Markovization

Order 1

- NP
- NNP
- NNP

Order \( \infty \)

- NP
- NNP
- NNP
- NNP

Symbols

<table>
<thead>
<tr>
<th>Horizontal Markov Order</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2v</td>
<td>2v</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>inf</td>
<td>inf</td>
</tr>
</tbody>
</table>

Horizontal Markov Order

- 74%
- 73%
- 72%
- 71%
- 70%
Vertical Markovization

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes. (cf. parent annotation)
Markovization

- This leads to a somewhat more general view of generative probabilistic models over trees

- Main goal: estimate \( P(\quad ) \)

- A bit of an interlude: *Tree-Insertion Grammars* deal with this problem more directly.
TIG: Insertion

\[ \phi \]

\[ \psi \]

\[ \phi' \]

\[ \psi \]

\[ S \]

\[ NP \downarrow \]

\[ VP \]

\[ V \]

\[ NP \downarrow \]

\[ saw \]

\[ NP \downarrow \]

\[ D \downarrow N \]

\[ man \]

\[ D \downarrow N \]

\[ V \]

\[ NP \downarrow \]

\[ man \]

\[ saw \]
Data-oriented parsing (Bod 1992)

- A case of *Tree-Insertion Grammars*
- Rewrite large (possibly lexicalized) subtrees in a single step
  
  ```
  S
  / \                      /
  NP  Aux  VP  PP
  /  \                  /  \                  /  \                  /  \
The post office  will hold out NP Conj NP as incentives
  / \                          / \                     / \
  discounts and service concessions
  ```

- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable *parse* is NP-complete due to unbounded number of “rules”
So the question is, how do we estimate these tree probabilities

- What type of tree-insertion grammar do we use?
- Equivalently, what type of independence assumptions do we impose?

Traditional PCFGs are only one type of answer to this question

```
S
  /
/  
NP VP
  /
PRP VBD ADJP
    /
He was right
```
Vertical and Horizontal

- Examples:
  - Raw treebank: \( v=1, h=\infty \)
  - Johnson 98: \( v=2, h=\infty \)
  - Collins 99: \( v=2, h=2 \)
  - Best F1: \( v=3, h=2v \)

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base: ( v=h=2v )</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
</tbody>
</table>
Unary Splits

- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.

- Solution: Mark unary rewrite sites with -U

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>
Tag Splits

• Problem: Treebank tags are too coarse.

• Example: Sentential, PP, and other prepositions are all marked IN.

• Partial Solution:
  • Subdivide the IN tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>
Other Tag Splits

- **UNARY-DT**: mark demonstratives as DT^U ("the X" vs. "those")
- **UNARY-RB**: mark phrasal adverbs as RB^U ("quickly" vs. "very")
- **TAG-PA**: mark tags with non-canonical parents ("not" is an RB^VP)
- **SPLIT-AUX**: mark auxiliary verbs with –AUX [cf. Charniak 97]
- **SPLIT-CC**: separate "but" and "&" from other conjunctions
- **SPLIT-%**: "%" gets its own tag.
Treebank Splits

- The treebank comes with some annotations (e.g., -LOC, -SUBJ, etc).
  - Whole set together hurt the baseline.
  - One in particular is very useful (NP-TMP) when pushed down to the head tag (why?).
  - Can mark gapped S nodes as well.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>81.8</td>
<td>9.3K</td>
</tr>
<tr>
<td>NP-TMP</td>
<td>82.2</td>
<td>9.6K</td>
</tr>
<tr>
<td>GAPPED-S</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
</tbody>
</table>
Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.

- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!

- Solution: annotate future elements into nodes.
  - Lexicalized grammars do this (in very careful ways – why?).

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>83.1</td>
<td>9.8K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
</tbody>
</table>
Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.
- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
  - Base NPs [cf. Collins 99]
  - Right-recursive NPs

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>86.0</td>
<td>11.7K</td>
</tr>
<tr>
<td>DOMINATES-V</td>
<td>86.9</td>
<td>14.1K</td>
</tr>
<tr>
<td>RIGHT-REC-NP</td>
<td>87.0</td>
<td>15.2K</td>
</tr>
</tbody>
</table>
This is panic buying.
Some Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Klein &amp; Manning 03</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>

- Beats “first generation” lexicalized parsers.
- Lots of room to improve – more complex models next.
Unlexicalized grammars: SOTA

- Klein & Manning 2003’s “symbol splits” were hand-coded
- Petrov and Klein (2007) used a hierarchical splitting process to learn symbol inventories
  - Reminiscent of decision trees/CART
- Coarse-to-fine parsing makes it very fast
- Performance is state of the art!

<table>
<thead>
<tr>
<th>Parser</th>
<th>≤ 40 words</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
</tr>
<tr>
<td>ENGLISH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charniak et al. (2005)</td>
<td>90.1</td>
<td>90.1</td>
</tr>
<tr>
<td>Petrov et al. (2006)</td>
<td>90.3</td>
<td>90.0</td>
</tr>
<tr>
<td>This Paper</td>
<td><strong>90.7</strong></td>
<td><strong>90.5</strong></td>
</tr>
<tr>
<td>ENGLISH (reranked)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charniak et al. (2005)</td>
<td><strong>92.4</strong></td>
<td><strong>91.6</strong></td>
</tr>
<tr>
<td>GERMAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dubey (2005)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This Paper</td>
<td><strong>80.8</strong></td>
<td><strong>80.7</strong></td>
</tr>
<tr>
<td>CHINESE&lt;sup&gt;5&lt;/sup&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chiang et al. (2002)&lt;sup&gt;b&lt;/sup&gt;</td>
<td>81.1</td>
<td>78.8</td>
</tr>
<tr>
<td>This Paper</td>
<td><strong>86.9</strong></td>
<td><strong>85.7</strong></td>
</tr>
</tbody>
</table>
Parse Reranking

- Nothing we’ve seen so far allows arbitrarily non-local features
- Assume the number of parses is very small
- We can represent each parse T as an arbitrary feature vector \( \phi(T) \)
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features
Parse Reranking

- Since the number of parses is no longer huge
  - Can enumerate all parses efficiently
  - Can use simple machine learning methods to score trees
  - E.g. maxent reranking: learn a binary classifier over trees where:
    - The top candidates are positive
    - All others are negative
    - Rank trees by $P(+) | T$)

- The best parsing numbers are from reranking systems
Shift-Reduce Parsers

- Another way to derive a tree:

- Parsing
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]
Derivational Representations

- Generative derivational models:
  \[ P(D) = \prod_{d_i \in D} P(d_i|d_0 \ldots d_{i-1}) \]

- How is a PCFG a generative derivational model?

- Distinction between parses and parse derivations.
  \[ P(T) = \sum_{D: D \rightarrow T} P(D) \]

- How could there be multiple derivations?
Tree-adjoining grammar (TAG)

- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don’t capture well (e.g. cross-serial dependencies)
TAG: Adjunction

\[
\begin{align*}
\phi & : A \\
\psi & : A^* \\
\end{align*}
\]
Recall that CFG parsing is $O(n^3)$

TAG parsing is $O(n^4)$

However, lexicalization causes the same kinds of complexity increases as in CFG
Combinatory Categorial Grammar
- Fully (mono-) lexicalized grammar
- Categories encode argument sequences
- Very closely related to the lambda calculus (more later)
- Can have spurious ambiguities (why?)

\[
\begin{align*}
John & \vdash \text{NP} \\
shares & \vdash \text{NP} \\
buys & \vdash (S\backslash\text{NP})/\text{NP} \\
sleeps & \vdash S\backslash\text{NP} \\
well & \vdash (S\backslash\text{NP}) \backslash (S\backslash\text{NP})
\end{align*}
\]
Digression: Is NL a CFG?

- Cross-serial dependencies in Dutch

... dat Wim Jan Marie de kinderen zag helpen leren zwemmen
... that Wim Jan Marie the children saw help teach swim

‘... that Wim saw Jan help Marie teach the children to swim’