Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L1: Automatic Linguistic Annotation
England’s fencers won gold on day 4 in Delhi with a medal-winning performance. This is Dr. Black’s second gold of the Games.
England’s fencers won gold on day 4 in Delhi with a medal-winning performance.

This is Dr. Black’s second gold of the Games.
Tokenisation (What’s a Word?)

England ’s fencers won gold on day 4 in Delhi with a medal- winning performance .

This is Dr. Black ’s second gold of the Games .
Part-of-Speech Tagging

England’s fencers won gold on day 4 in Delhi with a medal-winning performance.

This is Dr. Black’s second gold of the Games.
Syntactic Parsing - Phrase Structure

Taken from Dienes (2004)
From 1953 to 1955, 9.8 billion Kent cigarettes with the filters were sold, the company said.
Syntactic Parsing - Dependency Structure

A hearing is scheduled on the issue today.
Why is Parsing Difficult?

- Obtaining a wide-coverage grammar which can handle arbitrary real text is challenging
- Natural language is surprisingly AMBIGUOUS
Syntactic Ambiguity

S
  NP
    John
  VP
    V
      saw
    NP
      DT
        the
      N
        man
      P
        with
    PP
      DT
        the
      N
        telescope

S
  NP
    John
  VP
    V
      saw
    NP
      DT
        the
      N
        telescope

  PP
    DT
      the
    N
      man
    P
      with
    DT
      the
      N
        telescope
Syntactic Ambiguity: the problem is worse than you think
Syntactic Ambiguity: the problem is worse than you think
Syntactic Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses
- Put the block in the box on the table beside the chair 5 analyses
- Put the block in the box on the table beside the chair before the table 14 analyses
- Put the block in the box on the table beside the chair before the table in the kitchen 42 analyses
  - ... 132 analyses
  - ... 469 analyses
  - ... 1430 analyses
  - ... 4862 analyses
Interesting Ambiguity Examples

- The a are of I
- The cows are grazing in the meadow
- John saw Mary

examples from Abney (1996)
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L2: Introduction to Statistical Parsing
- Where does the grammar come from?
- What's the algorithm for generating possible parses?
- How do we decide between all the parses?

Taken from Dienes(2004)
The Penn Treebank

- Provides the possible phrase structure rules
- Provides data for estimating parse selection models
- Provides test data for evaluation

Taken from Dienes (2004)
Problems with the PTB Parsing Task

- It focuses on one language (English)
- It focuses on one domain (newswire)
- The test data hasn’t changed for 20 years
Currently the dominant parsing paradigm:

- there are treebanks for lots of languages
- dependencies are useful for various tasks
- performance is comparable to PS parsers
- the “grammar” is easy to understand (!)
- almost entirely data driven
Dependency Trees

taken from Wang and Zhang, NAACL tutorial 2010
Dependency Trees more Formally

- A directed graph with the following constraints:
  - connected
  - acyclic
  - single-head
  - projective (no crossing links)
Crossing Dependencies

- Rare in English, but more common in other languages (Czech, German)
- Requires a different parsing algorithm
Graph-Based Models

- Score each possible tree (according to some model)
- Search for the tree with the highest score
- Dynamic Programming (DP) typically used to do the (optimal) search

taken from Wang and Zhang, NAACL tutorial 2010
Edge-Based Factorisation Model

\[ Y^* = \arg \max_{Y \in \Phi(X)} \text{score}(Y|X) \]

\[ = \arg \max_{Y \in \Phi(X)} \sum_{x_i \rightarrow x_j \in Y} \text{score}(x_i \rightarrow x_j) \]

where \( X \) is the sentence, 
\( \Phi(X) \) is the set of possible dependency trees for \( X \), 
\( x_i \rightarrow x_j \) is a dependency link between words \( x_i \) and \( x_j \)
Edge-Based Linear Model

\[ \text{score}(x_i \rightarrow x_j) = \sum_k \lambda_k \cdot f_k(x_i \rightarrow x_j) \]

\[ = \bar{\lambda} \cdot \bar{f}(x_i \rightarrow x_j) \]

- Features have to be local to an edge in the graph
- but can span the whole sentence
- Various ways to estimate the weights, including structured perceptron
- Large numbers of binary features are needed for good performance
- but recent work using neural networks obviates this need
Example Features

Basic Features

I saw her duck with a telescope

- Uni-gram features
- Bi-gram features
- In between POS features
- Surrounding word POS features

taken from Wang and Zhang, NAACL tutorial 2010
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L3: Graph-Based Dependency Parsing
Untyped Dependency Trees

A tree is projective iff an edge from word w to word u implies that w is an ancestor of all words between w and u.

Taken from McDonald et al.
Edge-Based Linear Model

Basic Features

- Uni-gram features
- Bi-gram features
- In between POS features
- Surrounding word POS features

\[ \text{score}(x_i \rightarrow x_j) = \sum_k \lambda_k \cdot f_k(x_i \rightarrow x_j) \]

taken from Wang and Zhang, NAACL tutorial 2010
Dependency Parsing Formally

\[ s(x, y) = \sum_{(i,j) \in y} s(i, j) = \sum_{(i,j) \in y} w \cdot f(i, j) \]

- \( x \) is a sentence, \( y \) is a tree
- \((i,j)\) is an edge from \(i\)th word to \(j\)th word
- \( s \) is the scoring function
- \( f \) is the feature function, \( w \) is the weight vector
Maximum Spanning Trees

Assume we know the weight vector, \( \mathbf{w} \)

Consider the following directed graph for sentence \( \mathbf{x} \):

\[
G_\mathbf{x} = (V_\mathbf{x}, E_\mathbf{x}) \text{ where}
\]

\[
V_\mathbf{x} = \{ x_0 = \text{root}, x_1, \ldots, x_n \} \text{ and}
\] \[
E_\mathbf{x} = \{ (i, j) : x_i \neq x_j, x_i \in V_\mathbf{x}, x_j \in V_\mathbf{x} - \text{root} \}
\]

The highest-scoring (projective) dependency tree is equivalent to the (projective) maximum spanning tree
Decoding: finding the MST

The Chu-Liu-Edmonds algorithm (1965,67) finds the MST for non-projective trees; there is an $O(n^2)$ implementation.

For projective trees, the CKY algorithm can be adapted for dependency parsing to give an $O(n^5)$ algorithm.

There is a clever alternative chart-based algorithm from Eisner (1996) which runs in $O(n^3)$.
CKY-style Dependency Parsing

[ Mary ] → loves → [ [ the ] → girl ← [ outdoors ] ]

Slide thanks to Jason Eisner
Why CKY is $O(n^5)$ not $O(n^3)$

... advocate  
... hug

visiting relatives  
visiting relatives

Slide thanks to Jason Eisner
### Dependency Parsing Algorithms

<table>
<thead>
<tr>
<th>Name</th>
<th>Inventor</th>
<th>Projectivity</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKY-style chart parsing</td>
<td>Cocke–Younger–Kasami</td>
<td>Projective</td>
<td>$O(n^5)$</td>
</tr>
<tr>
<td>Eisner $O(n^3)$ parsing alg.</td>
<td>Eisner (96)</td>
<td>Projective</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Maximum Spanning Tree</td>
<td>Chu-Liu-Edmonds (65, 67)</td>
<td>Non-projective</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Shift-Reduce style parsing</td>
<td>Yamada, Nivre</td>
<td>Projective</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

taken from Wang and Zhang, NAACL tutorial 2010
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here

taken from Wang and Zhang, NAACL tutorial 2010
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here → S → He does it here
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here → S → He does it here → AL → He does it here

He
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here → S → He does it here → AL → does it here → S → does it here

He

He
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

He does it here \(\rightarrow\) S \(\rightarrow\) He does it here \(\rightarrow\) AL \(\rightarrow\) does it here \(\rightarrow\) S \(\rightarrow\) does it here

He

AR

does it here

He

does it here

He
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight

Diagram:

- He does it here → S
- He does it here → AL
- He does it here → S
- He does it here → AR
- He does it here → R
- He does it here → AR

Diagram sequence:

1. He does it here → S
2. He does it here → AL
3. He does it here → S
4. He does it here → AR
5. He does it here → R
6. He does it here → AR
Shift-Reduce Dependency Parsing

- S – Shift
- R – Reduce
- AL – ArcLeft
- AR – ArcRight
Greedy Local Search

Suffers from search errors, but potentially very fast (linear time)

taken from Wang and Zhang, NAACL tutorial 2010
Beam Search

Suffers from fewer search errors, but less fast (still linear time)
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L4: The Perceptron Parsing Model
Edge-Based Linear Model

Basic Features

- Uni-gram features
- Bi-gram features
- In between POS features
- Surrounding word POS features

\[
\text{score}(x_i \rightarrow x_j) = \sum_k \lambda_k \cdot f_k(x_i \rightarrow x_j)
\]

taken from Wang and Zhang, NAACL tutorial 2010
Features in the MST Parser

Table 1: Features used by system. p-word: word of parent in dependency edge. c-word: word of child. p-pos: POS of parent. c-pos: POS of child. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child.

<table>
<thead>
<tr>
<th>Basic Uni-gram Features</th>
<th>Basic Bi-gram Features</th>
<th>In Between POS Features</th>
<th>Surrounding Word POS Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-word, p-pos</td>
<td>p-word, p-pos, c-word, c-pos</td>
<td>p-pos, b-pos, c-pos</td>
<td>p-pos, p-pos+1, c-pos-1, c-pos</td>
</tr>
<tr>
<td>p-word</td>
<td>p-pos, c-word, c-pos</td>
<td>p-pos-1, p-pos, c-pos</td>
<td>p-pos, p-pos-1, c-pos-1, c-pos-1, c-pos</td>
</tr>
<tr>
<td>p-pos</td>
<td>p-word, p-pos, c-word</td>
<td>p-pos, p-pos+1, c-pos</td>
<td>p-pos, p-pos+1, c-pos, c-pos+1</td>
</tr>
<tr>
<td>c-word, c-pos</td>
<td>p-word, c-word</td>
<td>p-pos, p-pos-1, p-pos, c-pos</td>
<td>p-pos, p-pos-1, p-pos, c-pos, c-pos+1</td>
</tr>
<tr>
<td>c-word</td>
<td>p-word, p-pos, c-word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c-pos</td>
<td>p-pos, c-word</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

taken from McDonald et al.
Global Linear Model

\[
Score(\tau) = \sum_{x_i \rightarrow x_j \in \tau} \sum_k w_k \cdot f_k(x_i \rightarrow x_j)
\]

\[
= \sum_k w_k \cdot \sum_{x_i \rightarrow x_j \in \tau} f_k(x_i \rightarrow x_j)
\]

\[
= \sum_k w_k \cdot f_k(\tau)
\]

\[
= w \cdot F(\tau)
\]
Generic Online Learning

Training data: $T = \{(x_t, y_t)\}_{t=1}^{T}$

1. $w_0 = 0; \ v = 0; \ i = 0$
2. for $n : 1..N$
3. \ for $t : 1..T$
4. \ $w^{(i+1)} = \text{update} \ w^{(i)} \ \text{according to instance} \ (x_t, y_t)$
5. \ $v = v + w^{(i+1)}$
6. \ $i = i + 1$
7. \ $w = v/(N \times T)$

taken from McDonald et al.
The Perceptron Update

Given sentence $x_t$ and correct tree $y_t$

$$z_t = \arg \max_z w_{t-1} \cdot F(x_t, z)$$

$$w_t = w_{t-1} + F(x_t, y_t) - F(x_t, z_t)$$
# CoNLL Shared Task Data

<table>
<thead>
<tr>
<th>Multilingual</th>
<th>Ar</th>
<th>Ba</th>
<th>Ca</th>
<th>Ch</th>
<th>Cz</th>
<th>En</th>
<th>Gr</th>
<th>Hu</th>
<th>It</th>
<th>Tu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotation</td>
<td>d</td>
<td>d</td>
<td>c+f</td>
<td>c+f</td>
<td>d</td>
<td>c+f</td>
<td>d</td>
<td>c+f</td>
<td>c+f</td>
<td>d</td>
</tr>
<tr>
<td>Training data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tokens (k)</td>
<td>112</td>
<td>51</td>
<td>431</td>
<td>337</td>
<td>432</td>
<td>447</td>
<td>65</td>
<td>132</td>
<td>71</td>
<td>65</td>
</tr>
<tr>
<td>Sentences (k)</td>
<td>2.9</td>
<td>3.2</td>
<td>15.0</td>
<td>57.0</td>
<td>25.4</td>
<td>18.6</td>
<td>2.7</td>
<td>6.0</td>
<td>3.1</td>
<td>5.6</td>
</tr>
<tr>
<td>Tokens/sentence</td>
<td>38.3</td>
<td>15.8</td>
<td>28.8</td>
<td>5.9</td>
<td>17.0</td>
<td>24.0</td>
<td>24.2</td>
<td>21.8</td>
<td>22.9</td>
<td>11.6</td>
</tr>
<tr>
<td>LEMMA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No. CPOSTAG</td>
<td>15</td>
<td>25</td>
<td>17</td>
<td>13</td>
<td>12</td>
<td>31</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>No. POSTAG</td>
<td>21</td>
<td>64</td>
<td>54</td>
<td>294</td>
<td>59</td>
<td>45</td>
<td>38</td>
<td>43</td>
<td>28</td>
<td>31</td>
</tr>
<tr>
<td>No. FEATS</td>
<td>21</td>
<td>359</td>
<td>33</td>
<td>0</td>
<td>71</td>
<td>0</td>
<td>31</td>
<td>50</td>
<td>21</td>
<td>78</td>
</tr>
<tr>
<td>No. DEPREL</td>
<td>29</td>
<td>35</td>
<td>42</td>
<td>69</td>
<td>46</td>
<td>20</td>
<td>46</td>
<td>49</td>
<td>22</td>
<td>25</td>
</tr>
<tr>
<td>No. DEPREL H=0</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>% HEAD=0</td>
<td>8.7</td>
<td>9.7</td>
<td>3.5</td>
<td>16.9</td>
<td>11.6</td>
<td>4.2</td>
<td>8.3</td>
<td>4.6</td>
<td>5.4</td>
<td>12.8</td>
</tr>
<tr>
<td>% HEAD left</td>
<td>79.2</td>
<td>44.5</td>
<td>60.0</td>
<td>24.7</td>
<td>46.9</td>
<td>49.0</td>
<td>44.8</td>
<td>27.4</td>
<td>65.0</td>
<td>3.8</td>
</tr>
<tr>
<td>% HEAD right</td>
<td>12.1</td>
<td>45.8</td>
<td>36.5</td>
<td>58.4</td>
<td>41.5</td>
<td>46.9</td>
<td>46.9</td>
<td>68.0</td>
<td>29.6</td>
<td>83.4</td>
</tr>
<tr>
<td>HEAD=0/sentence</td>
<td>3.3</td>
<td>1.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.0</td>
<td>2.0</td>
<td>1.0</td>
<td>1.2</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>% Non-proj. arcs</td>
<td>0.4</td>
<td>2.9</td>
<td>0.1</td>
<td>0.0</td>
<td>1.9</td>
<td>0.3</td>
<td>1.1</td>
<td>2.9</td>
<td>0.5</td>
<td>5.5</td>
</tr>
<tr>
<td>% Non-proj. sent.</td>
<td>10.1</td>
<td>26.2</td>
<td>2.9</td>
<td>0.0</td>
<td>23.2</td>
<td>6.7</td>
<td>20.3</td>
<td>26.4</td>
<td>7.4</td>
<td>33.3</td>
</tr>
<tr>
<td>Punct. attached</td>
<td>S</td>
<td>S</td>
<td>A</td>
<td>S</td>
<td>S</td>
<td>A</td>
<td>S</td>
<td>A</td>
<td>A</td>
<td>S</td>
</tr>
<tr>
<td>DEPRELS for punct.</td>
<td>10</td>
<td>13</td>
<td>6</td>
<td>29</td>
<td>16</td>
<td>13</td>
<td>15</td>
<td>1</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

Taken from Nivre et al. (2007)
Graph-based vs. Transition-based

Table 1
Labeled parsing accuracy for top-scoring systems at CoNLL-X (Buchholz and Marsi 2006).

<table>
<thead>
<tr>
<th>Language</th>
<th>Graph-based (McDonald, Lerman, and Pereira 2006)</th>
<th>Transition-based (Nivre et al. 2006)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>66.91</td>
<td>66.71</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>87.57</td>
<td>87.41</td>
</tr>
<tr>
<td>Chinese</td>
<td>85.90</td>
<td>86.92</td>
</tr>
<tr>
<td>Czech</td>
<td>80.18</td>
<td>78.42</td>
</tr>
<tr>
<td>Danish</td>
<td>84.79</td>
<td>84.77</td>
</tr>
<tr>
<td>Dutch</td>
<td>79.19</td>
<td>78.59</td>
</tr>
<tr>
<td>German</td>
<td>87.34</td>
<td>85.82</td>
</tr>
<tr>
<td>Japanese</td>
<td>90.71</td>
<td>91.65</td>
</tr>
<tr>
<td>Portuguese</td>
<td>86.82</td>
<td>87.60</td>
</tr>
<tr>
<td>Slovene</td>
<td>73.44</td>
<td>70.30</td>
</tr>
<tr>
<td>Spanish</td>
<td>82.25</td>
<td>81.29</td>
</tr>
<tr>
<td>Swedish</td>
<td>82.55</td>
<td>84.58</td>
</tr>
<tr>
<td>Turkish</td>
<td>63.19</td>
<td>65.68</td>
</tr>
<tr>
<td>Average</td>
<td>80.83</td>
<td>80.75</td>
</tr>
</tbody>
</table>

Taken from McDonald & Nivre (2011)

Taken from Weiss et al. (2015)
### Accuracy League Table (2015)

Results for English WSJ

<table>
<thead>
<tr>
<th>Method</th>
<th>UAS</th>
<th>LAS</th>
<th>Beam</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bohnet (2010)</td>
<td>92.88</td>
<td>90.71</td>
<td>n/a</td>
</tr>
<tr>
<td>Martins et al. (2013)</td>
<td>92.89</td>
<td>90.55</td>
<td>n/a</td>
</tr>
<tr>
<td>Zhang and McDonald (2014)</td>
<td>93.22</td>
<td>91.02</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Transition-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*Zhang and Nivre (2011)</td>
<td>93.00</td>
<td>90.95</td>
<td>32</td>
</tr>
<tr>
<td>Bohnet and Kuhn (2012)</td>
<td>93.27</td>
<td>91.19</td>
<td>40</td>
</tr>
<tr>
<td>Chen and Manning (2014)</td>
<td>91.80</td>
<td>89.60</td>
<td>1</td>
</tr>
<tr>
<td>S-LSTM (Dyer et al., 2015)</td>
<td>93.20</td>
<td>90.90</td>
<td>1</td>
</tr>
<tr>
<td>Our Greedy</td>
<td>93.19</td>
<td>91.18</td>
<td>1</td>
</tr>
<tr>
<td>Our Perceptron</td>
<td><strong>93.99</strong></td>
<td><strong>92.05</strong></td>
<td>8</td>
</tr>
</tbody>
</table>

Taken from Weiss et al. (2015)
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L5: Categorial Grammar
Categorial Grammar (CG)

- Main responsibility for defining syntactic form is in the lexicon
- Hence CG is a lexicalized theory of grammar
  - along with other theories of grammar such as HPSG, TAG, LFG, ...
- Attractive linguistically because all language-dependent properties reside in the lexicon
  - small number of combination rules are language-invariant
- Also attractive computationally; e.g. supertagging for Categorial Grammar leads to highly efficient parsing (Clark and Curran, 2007)
Connection with Semantics

- Categorial Grammar has a strong commitment to Frege’s Principle of Compositionality (along with Montague from the 70s):
- The meaning of a phrase is a function of the meanings of the parts of the phrase and how those parts are put together
Contrast with Phrase-Structure Rules

- Early Chomskian approach and much work in Generative Grammar uses rewrite rules or productions (as in a Context Free Grammar):

  \[ S \rightarrow NP \ VP \]
  \[ VP \rightarrow TV \ NP \]
  \[ TV \rightarrow \{ \text{likes, sees,} \ldots \} \]

- Categorial Grammar captures the same information by assigning a functional type, or category, to grammatical entities

- Has roots in early work by Polish mathematician Ajdukiewicz (1935) and even earlier in Russel's theory of types
Lexical Categories

- An elementary syntactic structure – a *lexical category* – is assigned to each word in a sentence, eg:

  *walked*: $S \backslash NP$ ‘give me an NP to my left and I return a sentence’

- Think of the lexical category for a verb as a *function*: NP is the argument, S the result, and the slash indicates the direction of the argument
Lexical Categories

• Atomic categories: $S, N, NP, PP, \ldots$ (not many more)

• Complex categories are built recursively from atomic categories and slashes

• Example complex categories for verbs:
  – intransitive verb: $S \backslash NP \; walked$
  – transitive verb: $(S \backslash NP)/NP \; respected$
  – ditransitive verb: $((S \backslash NP)/NP)/NP \; gave$
A Simple CG Derivation

\[
\text{interleukin} - 10 \quad \text{inhibits} \quad \text{production} \\
\hline
NP \quad (S \backslash NP)/NP \quad NP
\]
A Simple CG Derivation

\[
\text{interleukin} - 10 \quad \text{inhibits} \quad \text{production} \\
\frac{NP}{(S\setminus NP)/NP} \quad NP \\
\frac{S\setminus NP}{NP} \\
\Rightarrow
\]

Forward application
A Simple CG Derivation

interleukin – 10  inhibits  production

\[
NP \quad \frac{(S\backslash NP)/NP}{NP} \quad NP \\
\]

\[
\frac{S\backslash NP}{NP} \\
S \quad <
\]

Backward application
Combination Rules in CG

- Can think of the categories in blue as “cancelling”
  - early work in CG talks about “cancellation rules”
- Also looks a bit like multiplication and division
- But fundamentally the lexical category for the verb is a function which is applied to its argument
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free
- So what is different to CFG?  
  – lexicalisation means that the information in CFG rewrite rules has been pushed down to the leaves of the derivation
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L6: Combinatory Categorial Grammar
Long-Range Dependencies

- A central problem for a theory of grammar:
  - “elements of sentences which belong together at the level of semantics or interpretation may be separated by unboundedly much intervening material” (Steedman)

- Obvious example in English is the relative clause construction:
  - a woman whom Warren likes
  - a woman whom Dexter thinks that Warren likes
  - …
The Relative Clause Construction

- Relative clause construction:
  - *a woman whom Warren likes*

  \[
  \begin{array}{cccc}
  a \text{ woman} & whom & Warren & \text{likes} \\
  NP & ? & NP & (S\backslash NP)/NP \\
  \end{array}
  \]

- *whom Warren likes* should be \(NP\backslash NP\)

- so *whom* should be \((NP\backslash NP)/X\) for some \(X\) to be determined
“Non-Constituents” in CCG

\[
a \text{woman} \quad \text{whom} \quad \text{Warren} \quad \text{likes} \\
NP \quad (NP\backslash NP)/X \quad NP \quad (S\backslash NP)/NP
\]

- Could \textit{Warren} \textit{likes} be a constituent?
- The coordination test for constituency suggests so:
  - \textit{Warren likes but Dexter detests contemporary dance}
- So what is its type?
  - how about \textit{S/NP}?
  - in which case the type of \textit{whom} is \((NP\backslash NP)/(S/NP)\)
Deriving “Non-Constituents”

- Can’t combine *Warren* and *likes* using application rules
- Need two new rules: type-raising and composition
Type-Raising

\[
\begin{array}{cccc}
& \text{a woman} & \text{whom} & \text{Warren} & \text{likes} \\
NP & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP & {S/(S\backslash NP)}^T \\
\end{array}
\]

- Subject \( NP \) becomes a functional category
- In general: \( NP \Rightarrow T/(T\backslash NP) \)
  - \( T \) is a variable; in practice, for both linguistic and practical parsing reasons, we’d want to limit \( T \) to a particular set of types
- Other categories can be type-raised, too, and we can have backward, as opposed to forward, type-raising
Forward Composition

\[
\begin{array}{cccc}
\text{a woman} & \text{whom} & \text{Warren} & \text{likes} \\
NP & (NP \setminus NP)/(S/NP) & NP & (S \setminus NP)/NP \\
& S/(S \setminus NP) & \xrightarrow{T} & S/NP \\
& \xrightarrow{B} \\
\end{array}
\]

- Composition allows us to "get inside" a functional category
- In general: \( X/Y \ Y/Z \Rightarrow X/Z \)
CCG Derivation for Relative Clause

\[
\begin{align*}
\text{NP} & \quad \frac{a\ \text{woman}}{(NP \backslash NP)/(S/\text{NP})} \quad \text{whom} \quad \frac{\text{Warren}}{NP} \quad \frac{\text{likes}}{(S \backslash NP)/\text{NP}} \\
& \quad \frac{S/(S \backslash NP)^T}{\text{S/\text{NP}}} \\
& \quad > \text{B} \\
& \quad \frac{\text{S/\text{NP}}}{\text{NP} \backslash \text{NP}} \\
& \quad \frac{\text{NP} \backslash \text{NP}}{\text{NP}}
\end{align*}
\]
“Spurious” Ambiguity

\[
\begin{align*}
Warren & \quad \text{likes} & \quad \text{the woman} \\
NP & \quad (S\setminus NP)/NP & \quad NP \\
S/(S\setminus NP) & \quad \rightarrow^T \\
S/NP & \quad \rightarrow^B \\
S & \quad \rightarrow
\end{align*}
\]

- Type-raising and composition can be used to analyse simple sentences with no long-range dependencies.
- A different derivation results, but the interpretation is the same (hence so-called “spurious ambiguity”).
Generalised Forward Composition

- Some linguistic phenomena suggest the need for additional combinatorial rules, eg:

  \[ I \text{ offered, and may give, a flower to a policeman} \]

- Need to coordinate \textit{offered} and \textit{may give}, which means we need to make \textit{may give} a constituent:

  \[
  (S\backslash NP)/(S\backslash NP) \ (S\backslash NP)/PP)/NP \Rightarrow \ ((S\backslash NP)/PP)/NP
  \]
Generalised Forward Composition

\[ X / Y \ (\ldots (Y / Z) / W) / \ldots \Rightarrow B^n (\ldots (X / Z) / W) / \ldots \]

- Can now combine *may* and *give*:

\[
\frac{\text{may}}{(S \backslash NP) / VP} \quad \frac{\text{give}}{(VP / PP) / NP} \quad \Rightarrow B^n
\]

\[
((S \backslash NP) / PP) / NP
\]

where \( VP = S \backslash NP \)
Argument Cluster Coordination

give a teacher an apple and a policeman a flower

• Looks like we need to coordinate *a teacher an apple* and *a policeman a flower*

• *Can a teacher an apple* really be a constituent?!

• Yes, if we allow backward type-raising and composition rules (once we allow these the derivation drops out)
Forward and Backward Type-Raising

\[ X \Rightarrow_T T/(T \setminus X) \quad \text{forward} \]
\[ X \Rightarrow_T T \setminus (T/X) \quad \text{backward} \]
Argument Cluster Coordination

\[
give \quad a \ teacher \quad an \ apple \quad and \quad a \ policeman \quad a \ flower
\]

\[
\begin{align*}
DTV & \quad NP & \quad NP & \quad \text{conj} & \quad NP \\
TV \backslash DTV & \quad VP \backslash TV & \quad TV \backslash DTV & \quad VP \backslash TV
\end{align*}
\]

where \( VP = S \backslash NP \), \( TV = (S \backslash NP) / NP \), \( DTV = ((S \backslash NP) / NP) / NP \)

- Now we need a rule to combine \( TV \backslash DTV \) and \( VP \backslash TV \)
Argument Cluster Coordination

\[
\begin{align*}
give & \quad \text{a teacher} \quad \text{an apple} \quad \text{and} \quad \text{a policeman} \quad \text{a flower} \\
\text{DTV} & \quad \text{NP} \quad \text{NP} \quad \text{conj} \quad \text{NP} \quad \text{NP} \\
\text{TV}\text{DTV} & \quad \text{VP}\text{TV} \quad <T \quad \text{VP}\text{TV} \quad <T \\
\text{VP}\text{DTV} & \quad <B\quad \text{VP}\text{DTV} \quad <B \\
\text{VP}\text{DTV} & \quad <\Phi\quad \text{VP}\text{DTV} \quad <\Phi \\
\text{VP} & \quad <\Phi \\
\text{VP} & \quad <\Phi \\
\end{align*}
\]

where \( VP = S\backslash NP \), \( TV = (S\backslash NP)/NP \), \( DTV = ((S\backslash NP)/NP)/NP \)

• Backward Composition \((< B)\):

\[
Y\backslash Z \quad X\backslash Y \quad \Rightarrow_B \quad X\backslash Z
\]
Backward Crossed Composition

I shall buy today and cook tomorrow some mushrooms

- buy today and cook tomorrow need to be constituents
- buy has category \((S\setminus NP)/NP\) and today has category \((S\setminus NP)/(S\setminus NP)\)
- No rule so far allows us to combine these; but this one will:

\[
Y/Z \ X\Y \Rightarrow_B \ X/Z \ (< B_x)
\]

\[
VP/NP \ VP\ VP \Rightarrow_B \ VP/NP
\]
Another Combinatory Rule

- Forward-Crossed Composition:

\[ X/Y \ Y\setminus Z \Rightarrow_{B_X} X\setminus Z \]

- Generalised Forward-Crossed Composition:

\[ X/Y (\ldots (Y\setminus Z)\setminus W)\ldots \Rightarrow_{B^n_X} (\ldots (X\setminus Z)\setminus W)\ldots \]

- Generalised case needed for the next derivation
- These rules not part of the English grammar
Cross-Serial Dependencies in Dutch

\[
\begin{align*}
S &< \quad S \setminus NP_1 \\
&< \quad (S \setminus NP_1) \setminus NP_2 \\
&< \quad (S \setminus NP_1) \setminus NP_3 \\
&< \quad (S \setminus NP_1) \setminus NP_4 \\
&< \quad (VP \setminus NP_3) \setminus NP_4 \\
&< \quad ((S \setminus NP_1) \setminus NP_2) \setminus NP_3 \\
&< \quad (((S \setminus NP_1) \setminus NP_2) \setminus NP_3) \setminus NP_4 \\
&< \quad ((((S \setminus NP_1) \setminus NP_2) \setminus NP_3) \setminus NP_4) / VP \\
&< \quad (VP \setminus NP_3) / VP \\
&< \quad (S \setminus NP_1) \setminus NP_2 \\
&< \quad (S \setminus NP_1) \setminus NP_3 \\
&< \quad (S \setminus NP_1) \setminus NP_4 \\
&< \quad NP_1 \\
&< \quad NP_2 \\
&< \quad NP_3 \\
&< \quad NP_4
\end{align*}
\]
Mild Context Sensitivity

- It is the generalised composition rules which lead to greater-than-context free power.
- A CCG with generalised composition and certain rule restrictions has the same generative power as Tree Adjoining Grammar (TAG) ("mildly context-sensitive").
- Interestingly, Kuhlman et al. show that relaxing some of the rule restrictions can provide a CCG with greater-than-context-free power, but with strictly less power than TAG.
Mild Context Sensitivity

Type 0 languages
Context sensitive languages
Context free languages
Regular languages

Mildly context sensitive languages = natural languages (?)
Introduction to Syntax and Parsing
ACS 2015/16
Stephen Clark
L7: A CCG Grammar and Treebank for naturally occurring text
Pierre Vinken, 61 years old, will join the board as a non-executive director Nov. 29.

Activation of the CD28 surface receptor provides a major costimulatory signal for T cell activation resulting in enhanced production of interleukin-2 (IL-2) and cell proliferation.

The Trust’s symbol, a sprig of oak leaves and acorns, is thought to have been inspired by a carving in the cornice of the Alfriston Clergy House.

- Can we really move from simple “linguistic” examples to sentences like these found in the real world?
Newspaper Example

Pierre | N/N Vinken | N, , 61 | N/N years | N old | (S[adj]\NP)\NP
, , will | (S[dcl]\NP)/(S[b]\NP) join | ((S[b]\NP)/PP)/NP
the | NP/N board | N as | PP/NP a | NP/N nonexecutive | N/N
director | N Nov. | ((S\NP)((S\NP)))/N 29 | N . |

- Needs an $N \rightarrow NP$ rule
- $S[adj]\NP$ is for predicative adjectives, e.g. the man is old
- We need a unary type-changing rule: $S[adj]\NP \rightarrow NP/NP$
- We need special rules in the parser to deal with punctuation
- Only need application in this example (no composition or type-raising)
Grammatical Features in CCGBank

- $S$ category often has a grammatical feature which indicates the kind of sentence or verb phrase
  - $S[dcl]$ declarative sentence
  - $S[q]$ yes/no questions
  - $S[b]$ bare infinitives
  - $S[to]$ to infinitives
  - $S[pss]$ past participles in passive mode
  - $S[pt]$ past participles in active mode
  - $S[ng]$ present participles
  - ...

- See p.47 of Julia’s thesis for full list

- $S$ in adverbial modifiers, e.g. $(S\backslash NP)/(S\backslash NP)$, effectively has a variable feature: $(S[X]\backslash NP)/(S[X]\backslash NP)$, which unifies with the feature on the argument and transfers to the result
Activation of (NP/NP)/NP the NP/N CD28 surface receptor provides (S[dcl]/NP)/NP a NP/N major costimulatory signal for (NP/NP)/NP T (N/N)/(N/N) cell activation resulting (S[ng]/NP)/PP in PP/NP enhanced production of (NP/NP)/NP interleukin-2 (IL-2) and conj cell proliferation.

- Needs a unary type-changing rule: $S[ng]/NP \rightarrow (S/NP)/(S/NP)$
- Need special rules to deal with brackets
- Still only needs application
Wikipedia Example

The Trust's symbol, a sprig of oak leaves and acorns, is thought to have been inspired by a carving in the cornice of Alfriston Clergy House.

- Still only need application
- No unary type-changing rules in this example
Unary Type-Changing Rules

- Without type-changing rules (notice that the category for used is non-standard and the category for once changes also):

<table>
<thead>
<tr>
<th>A form of asbestos</th>
<th>once</th>
<th>used</th>
<th>to make Kent cigarettes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(NP\NP)/(NP\NP)</td>
<td>(NP\NP)/(S[to]\NP)</td>
<td>S[to]\NP</td>
</tr>
</tbody>
</table>

- With type-changing rules (uses standard categories for used and once):

<table>
<thead>
<tr>
<th>A form of asbestos</th>
<th>once</th>
<th>used</th>
<th>to make Kent cigarettes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S\NP)/(S\NP)</td>
<td>(S[pss]\NP)/(S[to]\NP)</td>
<td>S[to]\NP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S[pss]\NP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NP\NP</td>
<td></td>
</tr>
</tbody>
</table>

- Type-changing rules increase the compactness of the lexicon (capturing generalisations) and reduce the number of categories assigned to modifiers such as once.
Real Examples using Composition

- Object extraction from a relative clause, using type-raising and forward composition:

```
That    finished    the job    that    Captain Chandler    had    begun
NP      (S[dcl]\NP)/NP  NP      (NP\NP)/(S[dcl]/NP)  NP      (S[dcl]/NP)/(S[pt]/NP)  (S[pt]/NP)/NP
```

- Question with an object extraction:

```
What    books    did    he    author    ?
(S[wq]/(S[q]/NP))/N  N       (S[q]/(S[b]/NP))/NP  NP      (S[b]/NP)/NP
```

Lots more real CCG data on my RESOURCES webpage
Creating a Treebank for CCG

- A CCG treebank consists of (sentence, CCG analysis) pairs
- The CCG analysis is likely to be a derivation, and may also contain additional information such as predicate-argument dependencies
- The treebank is useful for:
  - deriving a wide-coverage grammar (or extending an existing one)
  - inducing statistical disambiguation models
- How can we build a CCG treebank?
  - manually from scratch (or at least by correcting the output of an existing CCG parser)
  - by automatically transforming the analyses from an existing treebank (e.g. the Penn Treebank) into CCG derivations
- Manual creation of a treebank is expensive so we choose the 2nd option
The Penn Treebank

- 50k sentences/1M words of WSJ text annotated with phrase-structure (PS) trees
- How might we turn this into a CCG treebank?
- What information do we need in the PS trees?
  - head information
  - argument/adjunct distinction (so we can derive the CCG categories)
  - trace information/extracted arguments so we can analyse long-range dependencies
Example PTB Tree (with traces)
The Basic Translation Algorithm

- Ignoring long-range dependency/trace information, the basic algorithm is straightforward:
  - foreach tree $\tau$
    * determineConstituentTypes($\tau$)
    * makeBinary($\tau$)
    * assignCategories($\tau$)
Determining Constituent Type

- Constituent type is either head, complement or adjunct
- This information is not marked explicitly in the PTB, but can be inferred (using heuristic rules) based on:
  - function tags in the PTB, e.g. –SBJ (subject), –TMP (temporal modifier), –DIR (direction)
  - constituent label of a node and its parent (e.g. NP daughters of VPs are complements, unless they carry a function tag such as –LOC, –DIR, –TMP and so on)
- Appendix A of Collins’ thesis gives a list of the head rules
- See p.362 of H&S 2007 and Appendix A of CCGbank manual
Binarizing the Tree

- A PTB tree is not binarized, whereas a CCG derivation is
- Insert dummy nodes into the tree such that:
  - all children to the left of the head branch off in a right-branching tree
  - all children to the right of the head branch off in a left-branching tree
- Some PTB structures are very flat, e.g. compound noun phrases – in the compound noun case we just assume a right-branching structure (but see Vadas and Curran for inserting NP structure into the PTB)
- See p.362 of H&S 2007
Assigning Categories

- The root node
  - mapping from categories of root nodes of PTB trees to CCG categories, e.g. \( \{VP\} \rightarrow S \backslash NP, \{S, SINV, SQ\} \rightarrow S \)

- Head and complement
  - category of complement child defined by a similar mapping, e.g. \( \{NP\} \rightarrow NP, \{PP\} \rightarrow PP \)
  - category of the head is a function which takes the category of the complement as argument and returns the category of the parent node; direction of the slash is given by the position of the complement relative to the head

- Head and adjunct
  - given a parent category \( C \), the category of an adjunct child is \( C / C \) if the adjunct child is to the left of the head child (a premodifier), or \( C \backslash C \) if it is to the right (postmodifier)
Long-Range Dependencies

(NP-SBJ (NP Brooks Brothers))
(, ,)
(SBAR (WHNP-1 (WDT which))
(S (NP-SBJ NNP Marks))
(VP (VBD bought)
  (NP (–NONE– T*-1))
  (NP-TMP last year))))

• The co-indexed trace element T*-1 is crucial in assigning the correct categories
  – used as an indication of the presence of a direct object for the verb
  – used to assign the correct category to the Wh-pronoun (using a similar mechanism to GPSG’s “slash-passing”)

• p.57 of the CCGbank manual has a detailed example
Properties of CCGbank

- 99.4% of the sentences in the PTB are translated into CCG derivations
- Words with the most number of category types:

<table>
<thead>
<tr>
<th>Word</th>
<th>num cats</th>
<th>Freq</th>
<th>Word</th>
<th>num cats</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>130</td>
<td>4237</td>
<td>of</td>
<td>59</td>
<td>22782</td>
</tr>
<tr>
<td>is</td>
<td>109</td>
<td>6893</td>
<td>that</td>
<td>55</td>
<td>7951</td>
</tr>
<tr>
<td>to</td>
<td>98</td>
<td>22056</td>
<td>LRB</td>
<td>52</td>
<td>1140</td>
</tr>
<tr>
<td>than</td>
<td>90</td>
<td>1600</td>
<td>not</td>
<td>50</td>
<td>1288</td>
</tr>
<tr>
<td>in</td>
<td>79</td>
<td>15085</td>
<td>are</td>
<td>48</td>
<td>3662</td>
</tr>
<tr>
<td>–</td>
<td>67</td>
<td>2001</td>
<td>with</td>
<td>47</td>
<td>4214</td>
</tr>
<tr>
<td>'s</td>
<td>67</td>
<td>9249</td>
<td>so</td>
<td>47</td>
<td>620</td>
</tr>
<tr>
<td>for</td>
<td>66</td>
<td>7912</td>
<td>if</td>
<td>47</td>
<td>808</td>
</tr>
<tr>
<td>at</td>
<td>63</td>
<td>4313</td>
<td>on</td>
<td>46</td>
<td>5112</td>
</tr>
<tr>
<td>was</td>
<td>61</td>
<td>3875</td>
<td>from</td>
<td>46</td>
<td>4437</td>
</tr>
</tbody>
</table>
More Statistics

- Lexicon has 74,669 entries for 44,210 word types (929,552 tokens)
- Average number of lexical categories per token is 19.2
- 1,286 lexical category types in total
  - 439 categories occur only once
  - 556 categories occur 5 times or more
- Coverage on unseen data: lexicon contains correct categories for 94% of tokens in section 00
  - 3.8% due to unknown words
  - 2.2% known words but not with the relevant category
Inducing a Grammar from CCGbank

- Grammar (lexicon) can be read off the leaves of the trees
Chart Parsing with CCG

- **Stage 1**
  - Assign POS tags and lexical categories to words in the sentence
  - Use taggers to assign the POS tags and categories
    - based on standard Maximum Entropy tagging techniques

- **Stage 2**
  - Combine the categories using the combinatorial rules
  - Can use standard bottom-up CKY chart-parsing algorithm

- **Stage 3**
  - Find the highest scoring derivation according to some model
    - e.g. generative model, CRF, perceptron
  - Viterbi algorithm finds this efficiently
CCG Supertagging

He goes on the road with his piano

A bitter conflict with global implications

- Baseline tagging accuracy is $\approx 72\%$
  - baseline is to assign tag most frequently seen with word in training data, and assign $N$ to unseen words
- Baseline for Penn Treebank POS tagging is $\approx 90\%$
CCG Multitagging

- Per-word tagging accuracy is ≈ 92%
- Potentially assign more than one category to a word
  - assign all categories whose probability is within some factor $\beta$ of the highest probability category
- Accuracy is over 97% at only 1.4 categories per word
- Accuracy is now high enough to serve as a front-end to the parser
CKY Algorithm

chart[i][j] is a cell containing categories spanning words from i to i + j
initialise chart with categories of span 1 (lexical categories)

LOOP over span of result category (j = 2 to SENT_LENGTH)
    LOOP over start position of left combining category (i = 0 to SENT_LENGTH - j)
        LOOP over span of left combining category (k = 1 to j - 1)
            chart[i][j] += Combine(chart[i][k], chart[i + k][j - k])
Chart Parsing

- DP algorithms can be run over the packed representation
- The Viterbi algorithm finds the highest scoring derivation
Linear Parsing Model

\[ \text{Score}(d, S') = \sum_i \lambda_i f_i(d) = \bar{\lambda} \cdot \phi(d) \]

- Features are **counts** over \( d \)
  - root category of \( d \) (plus lexical head)
  - \(<\text{lexical category, lexical item}>\) pairs
  - rule feature: \( S \rightarrow NP \, S \backslash NP \) (plus lexical head)
  - predicate argument dependency: subj(bought, IBM) (plus distance)
  - “Back-calling” features with words replaced by POS tags
- Use Perceptron training to set the weights
Training Data from CCGbank

subj(persuades, Marks)
obj(persuades, Brooks)
subj(merge, Brooks)
to-inf(persuades, merge)
Feature Representation

$$f_i : D \rightarrow \mathcal{N} \quad (3000000 \leq i \leq 1)$$
Linear Parsing Model

\[
\text{Score}(d, s) = \sum_i \lambda_i f_i(d) = \bar{\lambda} \cdot \bar{f}(d)
\]

- \(f_i\) are the features (defined by hand)
- \(\lambda_i\) are the corresponding weights (which need to be learned)
Perceptron Training

\[ \text{Score}(d, S) = \sum_{i} \lambda_i f_i(d) = \bar{\lambda} \cdot \phi(d) \]

**Inputs:** training examples \((x_i, y_i)\)

**Initialisation:** set \(\bar{\lambda} = 0\)

**Algorithm:**

for \(t = 1..T, i = 1..N\)

1. calculate \(z_i = \arg \max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \bar{\lambda}\)
2. if \(z_i \neq y_i\)
   \[ \bar{\lambda} = \bar{\lambda} + \Phi(x_i, y_i) - \Phi(x_i, z_i) \]

**Outputs:** \(\bar{\lambda}\)
Perceptron Training
Perceptron Training
Perceptron Training

UPDATE WEIGHTS:

<table>
<thead>
<tr>
<th>SENT1:</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>w4</th>
<th>w5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S/NP)/PP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/(S/NP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/(S/NP)/PP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(NP)/(VP)/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(NP)/(VP)/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(NP)/(VP)/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(NP)/(VP)/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(NP)/(VP)/NP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

W1 = <0, 1, 0, ..., -1, 0, ..., -1, 0, ..., 0>

f1, f20, f55, f100, f210, f345
f19, f25, f75, f150, f211, f346, f450, f500, f525
f15, f21, f56, f120, f212, f348, f419
Perceptron Training

$W_1 = <0, 1, 0, ..., -1, 0, ..., -1, 0, 1, 0, -1, ..., 0>$
Perceptron Training

\[ W_1 = <0, 1, 0, ..., -1, 0, ..., -1, 0, 1, -1, ..., 0> \]

\[ f_{11}, f_{21}, f_{57}, f_{90}, f_{145}, f_{250} \]
\[ f_{21}, f_{25}, f_{76}, f_{151}, f_{222}, f_{348}, f_{444}, f_{507}, f_{575} \]
\[ f_{17}, f_{45}, f_{155}, f_{167}, f_{678} \]
Perceptron Training

W2 = <0, 2, -1, ..., 1, 1, ..., -1, 0, ..., 1, 0, -2, ..., -1>

f11, f21, f57, f90, f145, f250
f21, f25, f76, f151, f222, f348, f444, f507, f575
f17, f45, f155, f167, f678
DP vs. Beam Search

- DP requires the optimal sub-problem property
- For efficient parsing this restricts the feature set
- An alternative is to apply a beam to each cell
- Now no restrictions on the features
- Max-violation perceptron used for training
Parser Evaluation

- Compare output of the parser with a *gold standard*
- Exact match metric sometimes used but a little crude
- Partial match against a set of *grammatical relations* currently the method of choice
  - measures recovery of semantically important relations
  - relatively theory-neutral representation
Head-based GRs

- *She gave the present to Kim*
  - (ncsubj gave She_)
  - (dobj gave present)
  - (iobj gave to)
  - (dobj to Kim)
  - (det present the)

- *The company wants to wean itself away from expensive gimmicks*
  - (xcomp to wants wean)
  - (iobj wean from)
  - (ncmod prt wean away)
  - (dobj wean itself)
  - (dobj from gimmicks)
  - (ncmod _ gimmicks expensive)
  - ...

Mapping CCG Dependencies to GRs

- Argument slots in CCG dependencies are mapped to GRs

<table>
<thead>
<tr>
<th>CCG lexical category</th>
<th>arg slot</th>
<th>GR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(S[dcl]\backslash NP_1)/NP_2$</td>
<td>1</td>
<td>(nsubj %l %f)</td>
</tr>
<tr>
<td>$(S[dcl]\backslash NP_1)/NP_2$</td>
<td>2</td>
<td>(dobj %l %f)</td>
</tr>
<tr>
<td>$(NP\backslash NP_1)/NP_2$</td>
<td>1</td>
<td>(prep %f %l)</td>
</tr>
<tr>
<td>$(NP\backslash NP_1)/NP_2$</td>
<td>2</td>
<td>(pobj %l %f)</td>
</tr>
<tr>
<td>$NP[nb]/N_1$</td>
<td>1</td>
<td>(det %f %l)</td>
</tr>
</tbody>
</table>

- Mapping is many-to-many
Test Suite: DepBank

- 700 sentences of newspaper text manually annotated with GRSs
- Calculate precision and recall over GRSs

\[
\text{Prec} = \frac{\# \text{ correct}}{\# \text{ proposed by parser}} \quad \text{Rec} = \frac{\# \text{ correct}}{\# \text{ in gold standard}}
\]

\[
F\text{-score} = \frac{2PR}{P + R}
\]
# Parsing Accuracy

<table>
<thead>
<tr>
<th>Prec</th>
<th>Rec</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>84.1</td>
<td>82.8</td>
<td>83.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GR</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncsbj</td>
<td>79.6</td>
</tr>
<tr>
<td>dobj</td>
<td>87.7</td>
</tr>
<tr>
<td>obj2</td>
<td>66.7</td>
</tr>
<tr>
<td>iobj</td>
<td>73.4</td>
</tr>
<tr>
<td>clausal</td>
<td>75.0</td>
</tr>
<tr>
<td>ncmdb</td>
<td>76.1</td>
</tr>
<tr>
<td>aux</td>
<td>92.8</td>
</tr>
<tr>
<td>det</td>
<td>95.1</td>
</tr>
<tr>
<td>conj</td>
<td>77.5</td>
</tr>
</tbody>
</table>