Practical Linguistically Motivated Parsing

with Combinatory Categorial Grammar

Stephen Clark

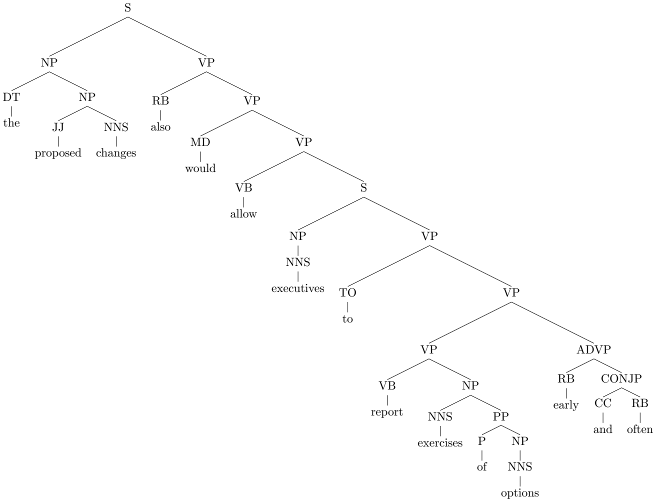
University of Cambridge Computer Laboratory

JHU Language Technology Summer School, June 2009

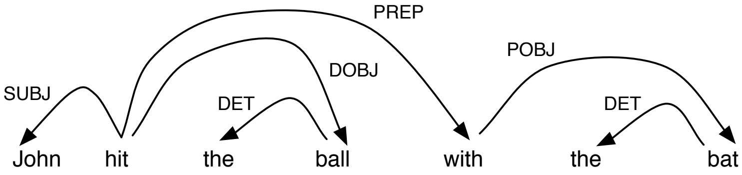
Natural Language Parsing

- Automatically assigning structure to a natural language input
- More specifically, taking a sentence as input and, using a pre-defined grammar, assigning some structure to it

Phrase Structure



Dependency Structure



Logical Form

```
From 1953 to 1955 , 9.8 billion Kent cigarettes with the filters were
sold , the company said .

  -------------   ------------------------------------------------------------------
 | x1          | | x2 x3                                                            |
 |-------------| |------------------------------------------------------------------|
(| company(x1) |A| say(x2)                                                          |)
 | single(x1)  | | agent(x2,x1)                                                     |
 |_____________| | theme(x2,x3)                                                     |
                 | proposition(x3)                                                  |
                 |                                                                  |
                 |        ------------------    ------------   ----------------     |
                 |       | x4               |  | x5         | | x6 x7 x8       |    |
                 |   x3: |------------------|  |------------| |----------------|    |
                 |      (| card(x4)=billion |;(| filter(x5) |A| with(x4,x5)    |))  |
                 |       | 9.8(x4)          |  | plural(x5) | | sell(x6)       |    |
                 |       | kent(x4)         |  |____________| | patient(x6,x4) |    |
                 |       | cigarette(x4)    |                 | 1953(x7)       |    |
                 |       | plural(x4)       |                 | single(x7)     |    |
                 |       |__________________|                 | 1955(x8)       |    |
                 |                                            | single(x8)     |    |
                 |                                            | to(x7,x8)      |    |
                 |                                            | from(x6,x7)    |    |
                 |                                            | event(x6)      |    |
                 |                                            |________________|    |
                 | event(x2)                                                        |
                 |__________________________________________________________________|
```

Why Build these Structures?

- We want to know the meaning of the sentence
- Structured representations allow us to access the semantics
- Who did What to Whom

Applications

- Question Answering/Semantic Search
- Machine Translation
- Information Extraction
- Dialogue Systems
- ...

Today's Tutorial

- Part I
  - why is automatic parsing difficult?
  - Combinatory Categorial Grammar
- Part II
  - parsing with CCG
  - statistical parsing models
  - parsing the web

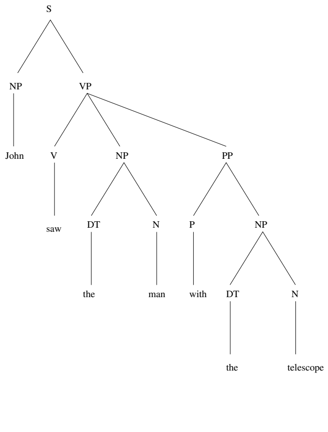
Why is Automatic Parsing Difficult?

- Obtaining a *wide-coverage* grammar which can handle arbitrary real text is challenging

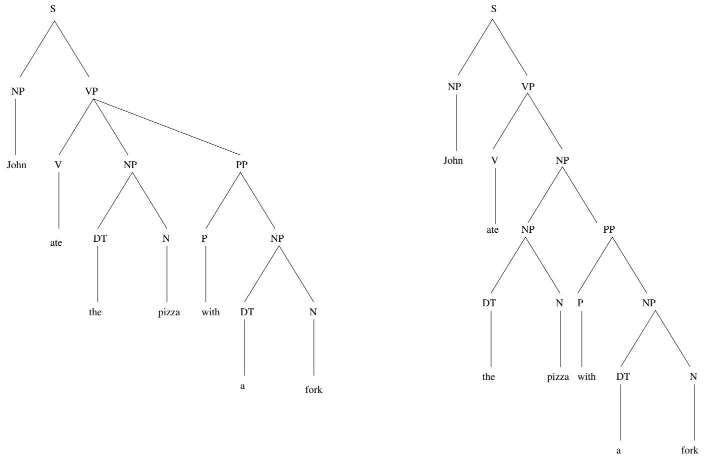
Why is Automatic Parsing Difficult?

- Obtaining a *wide-coverage* grammar which can handle arbitrary real text is challenging
- Natural language is surprisingly ambiguous

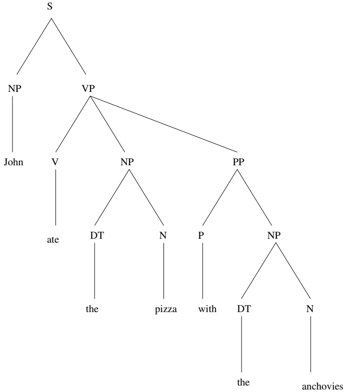
Syntactic Ambiguity

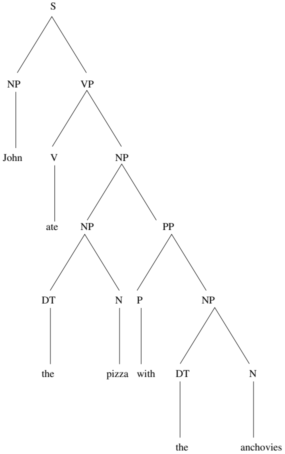


Ambiguity: the problem is worse than you think



Ambiguity: the problem is worse than you think





Ambiguity: the problem is even worse than that

- Put the block in the box on the table 2 analyses

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

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

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

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

Ambiguity: the problem is even worse than that

- Wider grammar coverage ⇒ more analyses
- In practice this could mean millions (or more) of parses for a single sentence
- We need a *parse model* giving the goodness of each parse
- We need an efficient representation of the large parse space, and an efficient way to search it

Grammars for Natural Language Parsing

- Standard approach is to use a Context Free Grammar

S $\longrightarrow$ NP VP
VP $\longrightarrow$ V NP, V NP PP
PP $\longrightarrow$ P NP
NP $\longrightarrow$ DT N
DT $\longrightarrow$ the, a
N $\longrightarrow$ cat, dog
V $\longrightarrow$ chased, jumped
P $\longrightarrow$ over

Combinatory Categorial Grammar (CCG)

- Categorial grammar (CG) is one of the oldest grammar formalisms (Ajdukiewicz, 1935; Bar-Hillel, 1953; Lambek 1958)
- Various flavours of CG now available: type-logical CG, algebraic pre-groups (Lambek), CCG
- CCG is now an established linguistic formalism (Steedman, 1996, 2000)
  - syntax; semantics; prosody and information structure; wide-coverage parsing; generation
  - http://groups.inf.ed.ac.uk/ccg/index.html

Combinatory Categorial Grammar (CCG)

- CCG is a lexicalised grammar
- An elementary syntactic structure – for CCG a lexical category – is assigned to each word in a sentence

walked: S\NP 'give me an NP to my left and I return a sentence'

Combinatory Categorial Grammar (CCG)

- CCG is a lexicalised grammar
- An elementary syntactic structure – for CCG a lexical category – is assigned to each word in a sentence

  walked: S\NP 'give me an NP to my left and I return a sentence'

- A small number of rules define how categories can combine
  – rules based on the *combinators* from Combinatory Logic

CCG Lexical Categories

- Atomic categories: S, N, NP, PP, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments

CCG Lexical Categories

- Atomic categories: S, N, NP, PP, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: $S\backslash NP$ *walked*
  - transitive verb: $(S\backslash NP)/NP$ *respected*
  - ditransitive verb: $((S\backslash NP)/NP)/NP$ *gave*

CCG Lexical Categories

- Atomic categories: S, N, NP, PP, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: $S\backslash NP$ *walked*
  - transitive verb: $(S\backslash NP)/NP$ *respected*
  - ditransitive verb: $((S\backslash NP)/NP)/NP$ *gave*
- Complex categories can encode modification
  - PP nominal: $(NP\backslash NP)/NP$
  - PP verbal: $((S\backslash NP)\backslash(S\backslash NP))/NP$

A Simple CCG Derivation

$$\dfrac{interleukin-10}{NP} \quad \dfrac{inhibits}{(S\backslash NP)/NP} \quad \dfrac{production}{NP}$$

A Simple CCG Derivation

$$\dfrac{\dfrac{interleukin-10}{NP} \quad \dfrac{\dfrac{inhibits}{(S\backslash NP)/NP} \quad \dfrac{production}{NP}}{S\backslash NP}>}{}$$

$>$ forward application

A Simple CCG Derivation

$$
\begin{array}{ccc}
\mathit{interleukin}-\mathit{10} & \mathit{inhibits} & \mathit{production} \\
\hline
NP & (S\backslash NP)/NP & NP \\
\cline{2-3}
 & \multicolumn{2}{c}{S\backslash NP \quad >} \\
\hline
\multicolumn{3}{c}{S \quad <}
\end{array}
$$

$>$ forward application

$<$ backward application

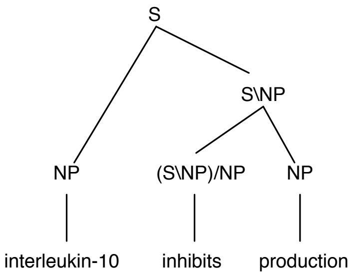
Function Application Rule Schemata

- Forward ($>$) and backward ($<$) application:

$$
\begin{array}{llcll}
X/Y & Y & \Rightarrow & X & (>) \\
Y & X\backslash Y & \Rightarrow & X & (<)
\end{array}
$$

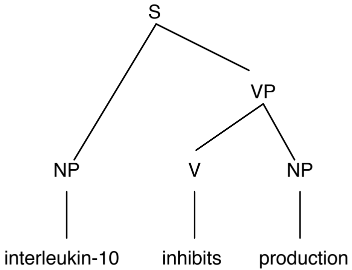
Classical Categorial Grammar

- 'Classical' Categorial Grammar only has application rules
- Classical Categorial Grammar is context free



Classical Categorial Grammar

- 'Classical' Categorial Grammar only has application rules
- Classical Categorial Grammar is context free



Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP
\end{array}
$$

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\hline NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
 & & & \overline{S/(S\backslash NP)}^{>\mathbf{T}} &
\end{array}
$$

$>\mathbf{T}$ type-raising

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\mathit{The} & \mathit{company} & \mathit{which} & \mathit{Microsoft} & \mathit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
\cline{4-4}
 & & & S/(S\backslash NP) \;{>}\mathbf{T} & \\
\cline{4-5}
 & & & \multicolumn{2}{c}{S/NP \;{>}\mathbf{B}}
\end{array}
$$

$>$ **T** type-raising

$>$ **B** forward composition

Extraction out of a Relative Clause

$$\begin{array}{ccccc}
\mathit{The} & \mathit{company} & \mathit{which} & \mathit{Microsoft} & \mathit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/\color{blue}{(S/NP)} & NP & (S\backslash NP)/NP
\end{array}$$

$$NP \xrightarrow{>\mathbf{T}} S/(S\backslash NP)$$

$$S/(S\backslash NP) \quad (S\backslash NP)/NP \xrightarrow{>\mathbf{B}} \color{blue}{S/NP}$$

$$(NP\backslash NP)/(S/NP) \quad S/NP \xrightarrow{>} NP\backslash NP$$

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
\multicolumn{2}{c}{\overline{\hphantom{NP/N\quad N}}>} & & \overline{\hphantom{NP}}>\mathbf{T} & \\
\multicolumn{2}{c}{NP} & & S/(S\backslash NP) & \\
 & & & \multicolumn{2}{c}{\overline{\hphantom{S/(S\backslash NP)\quad (S\backslash NP)/NP}}>\mathbf{B}} \\
 & & & \multicolumn{2}{c}{S/NP} \\
 & & \multicolumn{3}{c}{\overline{\hphantom{(NP\backslash NP)/(S/NP)\quad S/NP}}>} \\
 & & \multicolumn{3}{c}{NP\backslash NP} \\
\multicolumn{4}{c}{\overline{\hphantom{NP\quad NP\backslash NP}}<} & \\
\multicolumn{4}{c}{NP} &
\end{array}
$$

Forward Composition and Type-Raising

- Forward composition ($>_{\mathbf{B}}$):

$$X/Y \quad Y/Z \quad \Rightarrow \quad X/Z \qquad (>_{\mathbf{B}})$$

- Type-raising ($\mathbf{T}$):

$$\begin{aligned} X & \Rightarrow T/(T\backslash X) \qquad (>_{\mathbf{T}}) \\ X & \Rightarrow T\backslash(T/X) \qquad (<_{\mathbf{T}}) \end{aligned}$$

- Extra combinatory rules increase the weak generative power to mild context -sensitivity

"Non-constituents" in CCG – Right Node Raising

Google	*sells*	*but*	*Microsoft*	*buys*	*shares*
NP	$(S\backslash NP)/NP$	$conj$	NP	$(S\backslash NP)/NP$	NP
$\overline{S/(S\backslash NP)}^{>\mathbf{T}}$			$\overline{S/(S\backslash NP)}^{>\mathbf{T}}$		

$>\mathbf{T}$ type-raising

"Non-constituents" in CCG – Right Node Raising

$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & NP \\
\xrightarrow{}{>}\mathbf{T} & & & \xrightarrow{}{>}\mathbf{T} & & \\
S/(S\backslash NP) & & & S/(S\backslash NP) & & \\
\multicolumn{2}{c}{\xrightarrow{}{>}\mathbf{B}} & & \multicolumn{2}{c}{\xrightarrow{}{>}\mathbf{B}} & \\
\multicolumn{2}{c}{S/NP} & & \multicolumn{2}{c}{S/NP} &
\end{array}
$$

$>\mathbf{T}$ type-raising

$>\mathbf{B}$ forward composition

"Non-constituents" in CCG – Right Node Raising

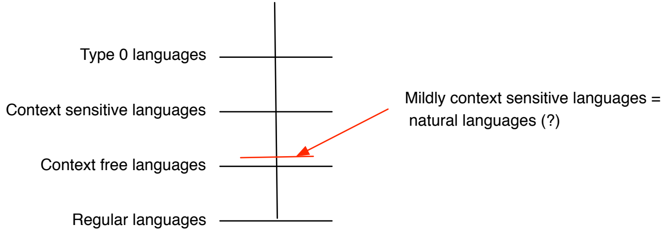
$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & NP \\
\xrightarrow{\quad\quad} >\mathbf{T} & & & \xrightarrow{\quad\quad} >\mathbf{T} & & \\
S/(S\backslash NP) & & & S/(S\backslash NP) & & \\
\multicolumn{2}{c}{\xrightarrow{\quad\quad\quad\quad} >\mathbf{B}} & & \multicolumn{2}{c}{\xrightarrow{\quad\quad\quad\quad} >\mathbf{B}} & \\
\multicolumn{2}{c}{S/NP} & & \multicolumn{2}{c}{S/NP} & \\
\multicolumn{5}{c}{\xrightarrow{\quad\quad\quad\quad\quad\quad\quad\quad} <\Phi>} & \\
\multicolumn{5}{c}{S/NP} &
\end{array}
$$

"Non-constituents" in CCG – Right Node Raising

$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & \color{blue}{NP} \\
\overline{S/(S\backslash NP)}\ {>}\mathbf{T} & & & \overline{S/(S\backslash NP)}\ {>}\mathbf{T} & & \\
\multicolumn{2}{c}{\overline{\qquad S/NP \qquad}\ {>}\mathbf{B}} & & \multicolumn{2}{c}{\overline{\qquad S/NP \qquad}\ {>}\mathbf{B}} & \\
\multicolumn{5}{c}{\overline{\qquad\qquad S/\color{blue}{NP} \qquad\qquad}\ {<}\Phi{>}} & \\
\multicolumn{6}{c}{\overline{\qquad\qquad\qquad S \qquad\qquad\qquad}\ {>}}
\end{array}
$$

Combinatory Categorial Grammar

- CCG is *mildly* context sensitive
- Natural language is provably non-context free
- Constructions in Dutch and Swiss German (Shieber, 1985) require more than context free power for their analysis
  - these have *crossing* dependencies (which CCG can handle)



Grammar Engineering vs. Grammar Extraction

- How can we obtain the wide-coverage grammar?
  - a syntactician writes the rules (whilst consulting corpus data)
  - a syntactician annotates sentences with grammatical structures, and the grammar is read automatically off that
  - the grammar is induced automatically from raw text

Grammar Engineering vs. Grammar Extraction

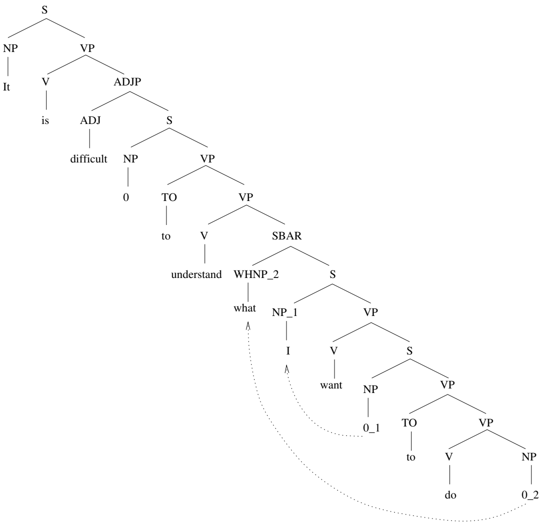
- How can we obtain the wide-coverage grammar?
  - a syntactician writes the rules (whilst consulting corpus data)
  - a syntactician annotates sentences with grammatical structures, and the grammar is read automatically off that
  - the grammar is induced automatically from raw text
- Introduces a level of modularity into the process:

linguist | computer scientist

The Penn Treebank (1993)

- 40,000 sentences (1M words) of English newspaper text annotated with phrase-structure trees
- Took annotators at the University of Pennsylvania 3 years to build
- Has been very influential (dominant) in parsing and NLP research

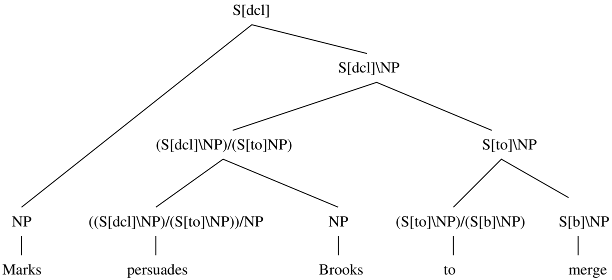
A PTB Phrase-Structure Tree



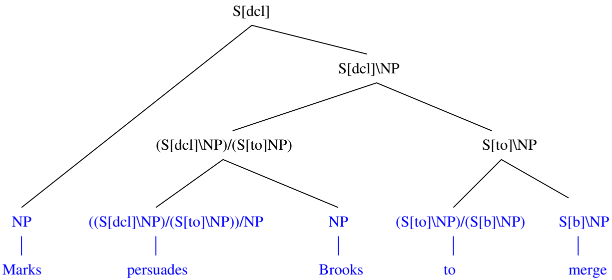
A CCG Treebank: CCGbank

- CCGbank developed by Hockenmaier and Steedman (Hockenmaier, 2003)
- Phrase-structure trees in Penn Treebank (semi-)automatically converted into CCG derivations
- But note phrase-structure trees not isomorphic to CCG analyses (e.g. coordination)

A CCG Derivation Tree



Inducing a Grammar



- Grammar (lexicon) can be read off the leaves of the trees
- In addition to the grammar, CCGbank provides training data for the statistical models

Inducing a Grammar

- $\approx$ 1 200 lexical category types in CCGbank (compared with 45 POS tags in Penn Treebank)
- Frequency cut-off of 10 gives $\approx$ 400 types (when applied to sections 2-21 of CCGbank)
  - this set has very high coverage on unseen data (section 00)
- In addition to the grammar, CCGbank provides training data for the statistical models

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

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 combinatory rules
  - Can use standard bottom-up CKY chart-parsing algorithm

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 combinatory 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 efficently

Maximum Entropy Tagging

BELL|NNP|N/N INDUSTRIES|NNP|N/N Inc.|NNP|N increased|VBD|$(S[dcl]\backslash NP)/NP$ its|PRP$|$NP[nb]/N$ quarterly|NN|N to|TO|$((S\backslash NP)\backslash(S\backslash NP))/NP$ 10|CD|N/N cents|NNS|N from|IN|$((S\backslash NP)\backslash(S\backslash NP))/NP$ seven|CD|N/N cents|NNS|N a|DT|$(NP\backslash NP)/N$ share|NN|N .|.|.

- Consider POS tagging as an example
- 45 POS tags from the Penn Treebank

Maximum Entropy Tagging (Ratnaparkhi, 1998)

- Use local log-linear models to estimate $P(\mathit{tag}|\mathit{context})$:

$$P(t|x) = \frac{1}{Z_x} e^{\sum_j \lambda_j f_j(t,x)}$$

Z_x is a normalisation constant ensuring a proper distribution

- Conditional probability of tag sequence:

$$P(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(t_i | x_i)$$

Feature-Based Tagging

- Context is a 5-word window surrounding target word
- Features are the words in the window, plus the two previously assigned tags
- Additional features for rare and unknown words
    - suffix information
    - is the word capitalised?
    - does the word contain a hyphen?

Features in Log-Linear Tagging Models

- Features are binary-valued indicator functions
- Contextual predicates identify elements of the context which may be useful for predicting the tag

$$f_i(t,x) = \begin{cases} 1 & \text{if } \texttt{word}(x) = \texttt{the} \;\; \& \;\; t = \text{det} \\ 0 & \text{otherwise} \end{cases}$$

- $\texttt{word}(x) = \texttt{the}$ is an example of a contextual predicate
- Features can be arbitrary properties of the context
- No requirement for the features to be independent
- Variety of training algorithms available to automatically set the weights

CCG Supertagging

$$\frac{He}{NP} \quad \frac{goes}{(S[dcl]\backslash NP)/PP} \quad \frac{on}{PP/NP} \quad \frac{the}{NP/N} \quad \frac{road}{N} \quad \frac{with}{((S\backslash NP)\backslash(S\backslash NP))/NP} \quad \frac{his}{NP/N} \quad \frac{piano}{N}$$

$$\frac{A}{NP/N} \quad \frac{bitter}{N/N} \quad \frac{conflict}{N} \quad \frac{with}{(NP\backslash NP)/NP} \quad \frac{global}{N/N} \quad \frac{implications}{N}$$

CCG Supertagging

He	*goes*	*on*	*the*	*road*	*with*	*his*	*piano*
NP	$(S[dcl]\backslash NP)/PP$	PP/NP	NP/N	N	$((S\backslash NP)\backslash(S\backslash NP))/NP$	NP/N	N

A	*bitter*	*conflict*	*with*	*global*	*implications*
NP/N	N/N	N	$(NP\backslash NP)/NP$	N/N	N

- $\approx$ 400 lexical category types
- 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%

Lexical Category Sequence for Newspaper Sentence

$\dfrac{In|IN}{(S/S)/NP}$ $\dfrac{an|DT}{NP[nb]/N}$ $\dfrac{Oct.|NNP}{N/N}$ $\dfrac{19|CD}{N/N}$ $\dfrac{review|NN}{N}$ $\dfrac{of|IN}{(NP\backslash NP)/NP}$ $\dfrac{The|DT}{NP[nb]/N}$

$\dfrac{Misanthrope|NNP}{N}$ $\dfrac{at|IN}{(NP\backslash NP)/NP}$ $\dfrac{Chicago|NNP}{N}$ $\dfrac{'s|POS}{(NP[nb]/N)\backslash NP}$ $\dfrac{Goodman|NNP}{N/N}$ $\dfrac{Theatre|NNP}{N}$

$\dfrac{-LRB-|LRB}{(NP\backslash NP)/S[dcl]}$ $\dfrac{Revitalized|JJ}{N/N}$ $\dfrac{Classics|NNS}{N}$ $\dfrac{Take|VBZ}{(S[dcl]\backslash NP)/NP}$ $\dfrac{the|DT}{NP[nb]/N}$ $\dfrac{Stage|NN}{N}$ $\ldots$

A Maximum Entropy Supertagger

- Maximum Entropy tagging method can be applied to CCG supertagging
- Features are the words and POS tags in the 5-word window, plus the two previously assigned categories
- Per-word tagging accuracy is $\approx 92\%$
- This accuracy is not high enough for the tagger to serve as an effective front-end to a CCG parser
  – roughly two errors per WSJ sentence on average

Multitagging

- Potentially assign more than one category to a word
  - assign all categories whose probability is within some factor β 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

Chart Parsing

- A chart is just a tabular data structure which stores the constituents spanning each subsequence of words
- The chart can be filled in “bottom-up”
  - start by combining lexical categories and continue to apply the combinatory rules until the whole sentence is covered
- Fill in the cells corresponding to the shortest subsequences first:
  - the *CKY algorithm*

Chart Parsing

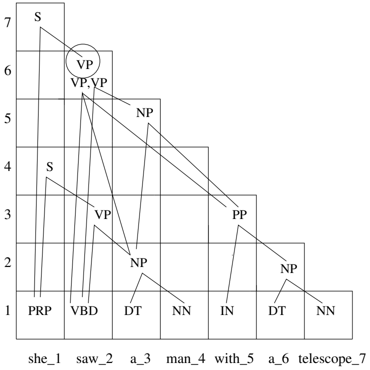
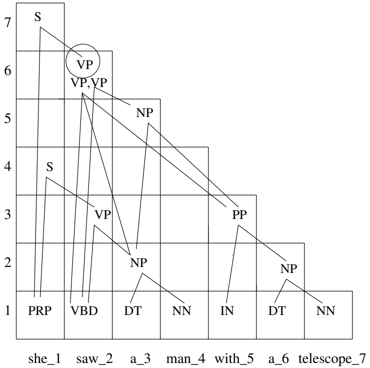
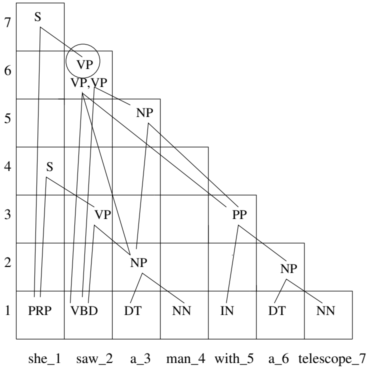


Chart Parsing



- CKY chart-parsing algorithm operates bottom-up

Chart Parsing



- CKY chart-parsing algorithm operates bottom-up
- *Packing* the chart efficiently represents a large derivation space

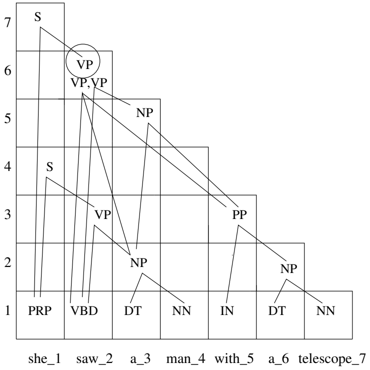
CKY Algorithm

```
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



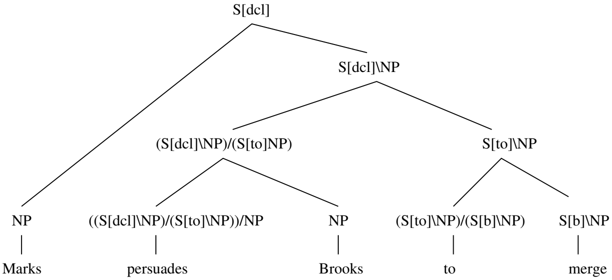
- DP algorithms can be run over the packed representation
- The *Viterbi* algorithm finds the highest scoring derivation

Linear Parsing Model

$$\text{Score}(d, S) = \sum_i \lambda_i f_i(d) = \overline{\lambda} \cdot \phi(d)$$

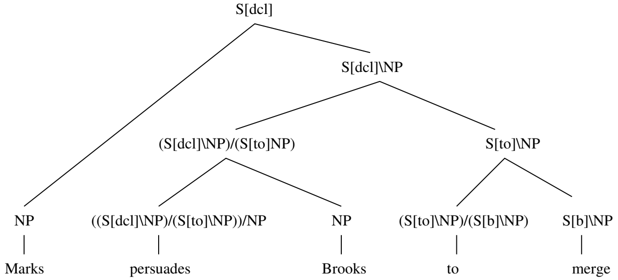
- Features are counts over d
  - root category of d (plus lexical head)
  - $\langle$lexical category, lexical item$\rangle$ pairs
  - rule feature: $S \rightarrow NP \;\; S\backslash NP$ (plus lexical head)
  - predicate argument dependency: subj(bought, IBM) (plus distance)
  - "Backing-off" features 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



$$f_i : D \to \mathcal{N} \qquad (3\,000\,000 \le i \le 1)$$

Linear Parsing Model

$$\text{Score}(d, s) = \sum_i \lambda_i . f_i(d) = \overline{\lambda} \cdot \overline{f}(d)$$

- f_i are the *features* (defined by hand)
- λ_i are the corresponding *weights* (which need to be learned)

Perceptron Training

$$\text{Score}(d, S) = \sum_i \lambda_i f_i(d) = \overline{\lambda} \cdot \phi(d)$$

Inputs: training examples (x_i, y_i)
Initialisation: set $\overline{\lambda} = 0$
Algorithm:
  for $t = 1..T$, $i = 1..N$
    calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \overline{\lambda}$
    if $z_i \neq y_i$
      $\overline{\lambda} = \overline{\lambda} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Outputs: $\overline{\lambda}$

Perceptron Training

W0 = <0,0,0,...,0,0,...,0,...0,0,0,0,...,0>

5	S S\NP S/S				
4	S/S S/NP	S\NP S\S (S\NP)\NP			
3	S	S\NP S/NP	NP VP\VP		
2	S/NP S/S	S\NP (S\NP)/PP	PP NP\NP	NP\NP VP\VP	
1	NP N S/(S\NP) (S/S)/NP	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP	NP N
SENT1:	w1	w2	w3	w4	w5

Perceptron Training

DECODE:

5	S S\NP S/S				
4	S/S S/NP	S\NP S\S (S\NP)\NP			
3	S	S\NP S/NP	NP VP\VP		
2	S/NP S/S	S\NP (S\NP)/PP	PP NP\NP	NP\NP VP\VP	
1	NP N S/(S\NP) (S/S)/NP	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP	NP N
SENT1:	w1	w2	w3	w4	w5

W0 = <0,0,0,...,0,0,...,0,...0,0,0,0,...,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 (Online)

UPDATE WEIGHTS:

5	S S\NP S/S				
4	S/S S/NP	S\NP S\S (S\NP)\NP			
3	S	S\NP S/NP	NP VP\VP		
2	S/NP S/S	S\NP (S\NP)/PP	PP NP\NP	NP\NP VP\VP	
1	NP N S/(S\NP) (S/S)/NP	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP	NP N
SENT1:	w1	w2	w3	w4	w5

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

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

4	S/S S/NP S			
3	S	S\NP PP/NP		
2	S/NP S/S	S\NP (S\NP)/PP	PP PP/NP NP\NP	
1	NP N S/(S\NP)	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP NP
SENT2:	w1	w2	w3	w4

Perceptron Training

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

DECODE:

4	S/S S/NP S			
3	S	S\NP PP/NP		
2	S/NP S/S	S\NP (S\NP)/PP	PP PP/NP NP\NP	
1	NP N S/(S\NP)	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP NP
SENT2:	w1	w2	w3	w4

f11, f21, f57, f90, f145, f250

f21, f25, f76, f151, f222, f348, f444, f507, f575

f17, f45, f155, f167, f678

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

UPDATE WEIGHTS:

4	S/S S/NP S			
3	S	S\NP PP/NP		
2	S/NP S/S	S\NP (S\NP)/PP	PP PP/NP NP\NP	
1	NP N S/(S\NP)	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP NP
SENT2:	w1	w2	w3	w4

Perceptron Training is Expensive

$$\text{Score}(d|S) = \sum_i \lambda_i f_i(d) = \overline{\lambda} \cdot \phi(d)$$

Inputs: training examples (x_i, y_i)
Initialisation: set $\overline{\lambda} = 0$
Algorithm:
for $t = 1..T$, $i = 1..N$
calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \overline{\lambda}$
if $z_i \neq y_i$
$\overline{\lambda} = \overline{\lambda} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Outputs: $\overline{\lambda}$

- Requires an efficient decoder

Efficient Decoding with CCG

- Supertagging leaves decoder with (relatively) little left to do
- Each packed chart needs at most 20 MB RAM
- Most probable derivation can be found very quickly with Viterbi
- Training takes 5 hours for 10 iterations

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 Grammatical Relations

- *She gave the present to Kim*

```
(ncsubj gave She _)
(dobj gave present)
(iobj gave to)
(dobj to Kim)
(det present the)
```

Head-Based Grammatical Relations

- *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

CCG lexical category	arg slot	GR
$(S[dcl]\backslash NP_1)/NP_2$	1	`(nsubj %l %f)`
$(S[dcl]\backslash NP_1)/NP_2$	2	`(dobj %l %f)`
$(NP\backslash NP_1)/NP_2$	1	`(prep %f %l)`
$(NP\backslash NP_1)/NP_2$	2	`(pobj %l %f)`
$NP[nb]/N_1$	1	`(det %f %l)`

- Mapping is many-to-many

Test Suite: DepBank

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

$$\textit{Prec} = \frac{\#\ \textit{correct}}{\#\ \textit{proposed by parser}} \qquad \textit{Rec} = \frac{\#\ \textit{correct}}{\#\ \textit{in gold standard}}$$

$$\textit{F-score} = \frac{2\,P\,R}{P+R}$$

Final Parsing Results

Prec	Rec	F-score
84.1	82.8	83.4

- These scores compare favourably with the best results in the literature on this test set

Results by Dependency Type

GR	F-score
ncsubj	79.6
dobj	87.7
obj2	66.7
iobj	73.4
clausal	75.0
ncmod	76.1
aux	92.8
det	95.1
conj	77.5

Parsing the Web

- Why parse the Web?
  - semantic search
  - provide massive amounts of data for knowledge acquisition
  - ...
- Need a fast parser (to process billions of web pages)
- Need a parser that isn't overly tuned to newspaper text

Speed Demo

- Use of the CCG supertagger (and some highly engineered C++) leads to a highly efficient linguistically motivated parser
- Can process **1 billion words** in less than 5 days with 18 machines
- Can we make the parser go faster still?

Conclusion

- Robust linguistically-motivated parsing of real text is now possible
  - but can it really help in NLP applications?
- What's left?
  - plenty of room for accuracy improvements
  - cheap ways to get more training data