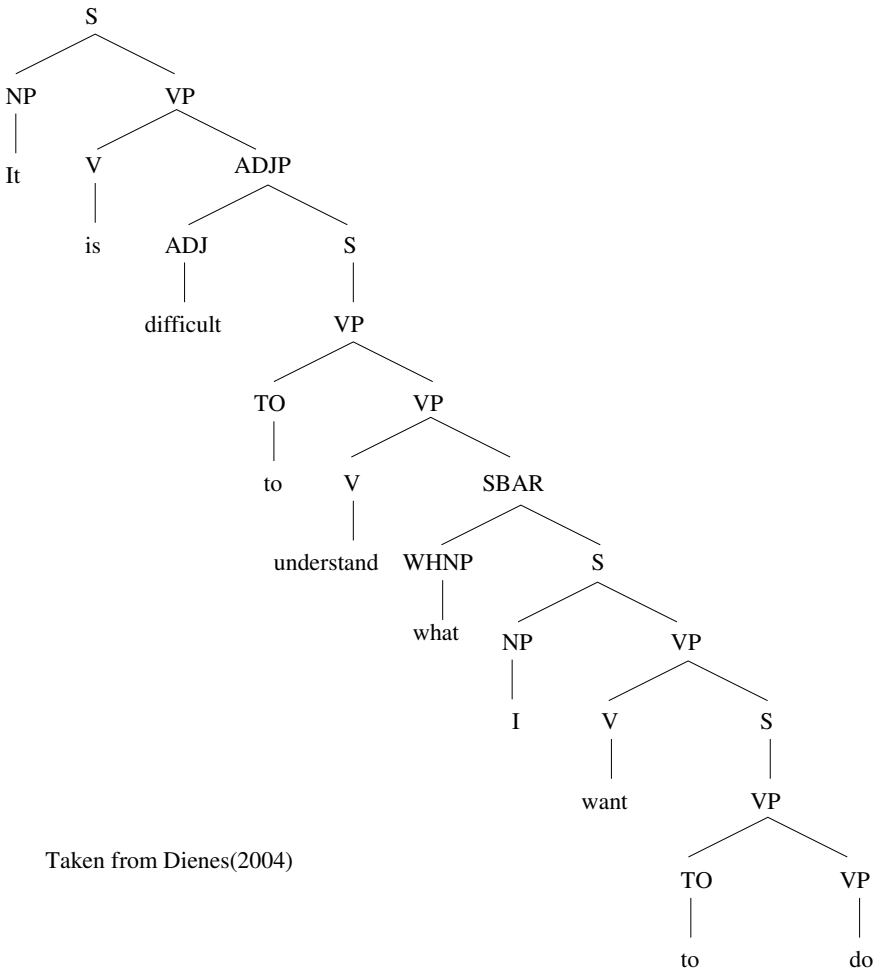
Linguistically Motivated Large-Scale Language Processing

Stephen Clark
(with James R. Curran and Mark Steedman)
stephen.clark@comlab.ox.ac.uk

Oxford University Computing Laboratory

CLUK/CamLing, Cambridge 2007

Typical Output of a Statistical Parser



Taken from Dienes(2004)

Parsing with the Penn Treebank (Marcus et al., 1993)

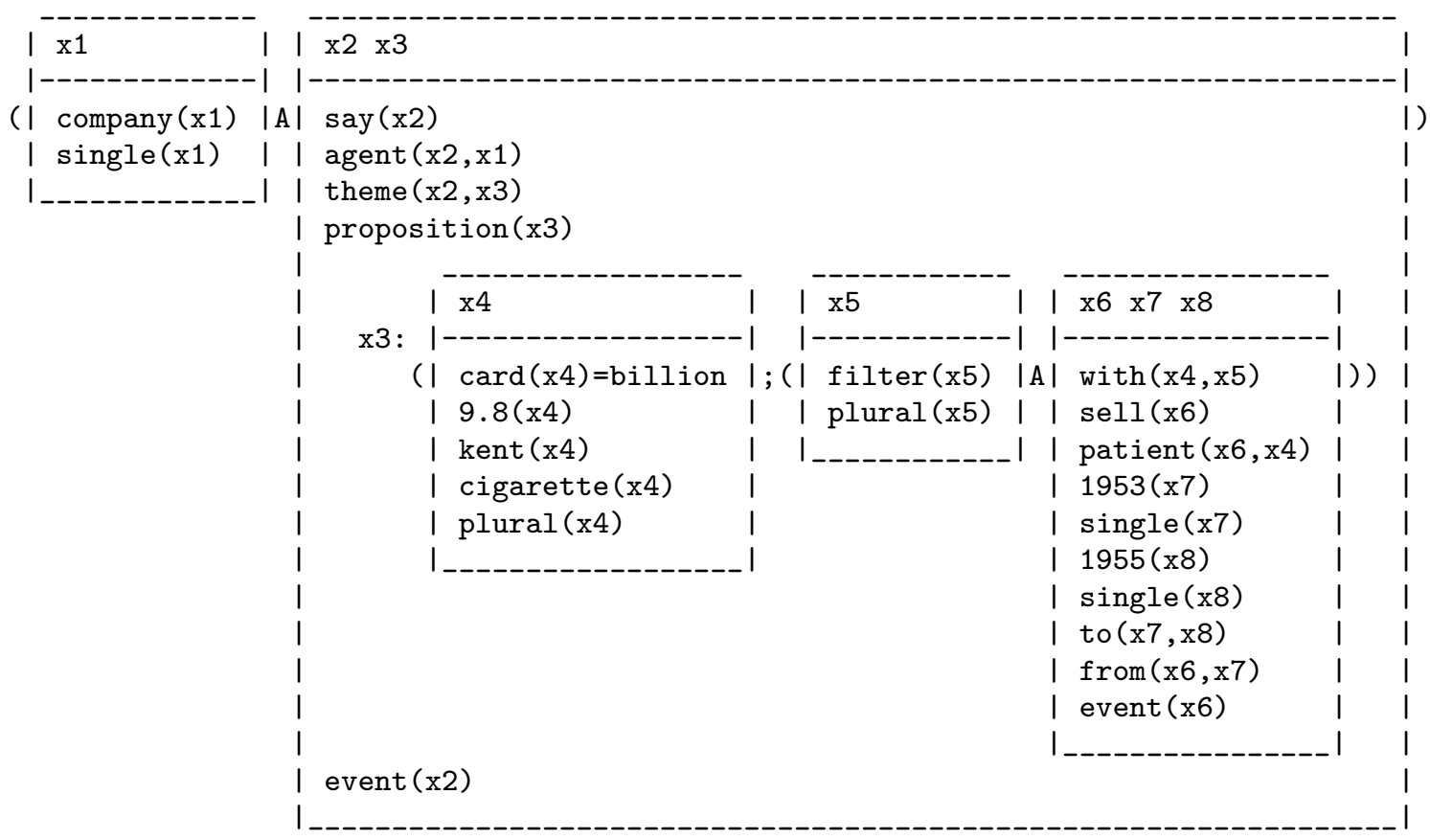
- Largest treebank available for English
- 40,000 sentences of mostly newspaper text (WSJ) each manually annotated with a phrase-structure tree
- (head-lexicalised) Context-free grammar rules can be read off the treebank
- Probabilities of rules can be estimated from the treebank (using relative frequencies)
- Search algorithm (eg Viterbi + beam) can find most probable tree for a new sentence

Linguistically Motivated Statistical Parsing

- Key benefits from statistical corpus-based approach are robustness and efficiency
- Accuracy is improving
  (with more sophisticated Machine Learning, eg CRFs, Perceptron)
- Recent development is the combination of statistical methods and grammar formalisms such as LFG, HPSG, TAG, and CCG

Towards Richer Output (Bos, 2005)

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



Statistical Parsing with CCG

- CCG is a lexicalised grammar formalism
- CCG was designed to handle the long-range dependencies in, e.g., extraction and coordination
- Transparent interface between surface syntax and underlying semantic representation

Statistical Parsing with CCG

- CCG is a lexicalised grammar formalism
- CCG was designed to handle the long-range dependencies in, e.g., extraction and coordination
- Transparent interface between surface syntax and underlying semantic representation
- But surely we sacrifice efficiency and robustness?
  - robustness obtained from using statistical methods and a grammar extracted from a CCG treebank
  - (surprising) efficiency obtained from exploiting lexicalised nature of the formalism (supertagging)

Outline

- Combinatory Categorial Grammar (CCG)
- Parsing with CCG
- Perceptron parsing model
- Parser evaluation on DepBank

Combinatory Categorial Grammar (CCG)

- Categorial grammar is one of the oldest grammar formalisms (Ajdukiewicz, 1935; Bar-Hillel, 1953)
- Classical categorial grammar is context free
- CCG is *mildly context-sensitive* (Vijay-Shanker & Weir, 1994) – weakly equivalent to Tree Adjoining Grammar
- CCG can analyse crossing dependencies in Dutch (non context-free)

Lexical Categories

- The *category* is the basic grammatical unit in CCG
  lexical categories are the categories assigned to words
- 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

Lexical Categories

- The *category* is the basic grammatical unit in CCG
  lexical categories are the categories assigned to words
- 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 can encode subcategorisation information
  - transitive verb: $(S\backslash NP)/NP$

Lexical Categories

- The *category* is the basic grammatical unit in CCG
  lexical categories are the categories assigned to words
- 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 can encode subcategorisation information
  - transitive verb: $(S\backslash NP)/NP$
- Complex categories can encode modification
  - PP-nominal: $(NP\backslash NP)/NP$

A CCG Derivation

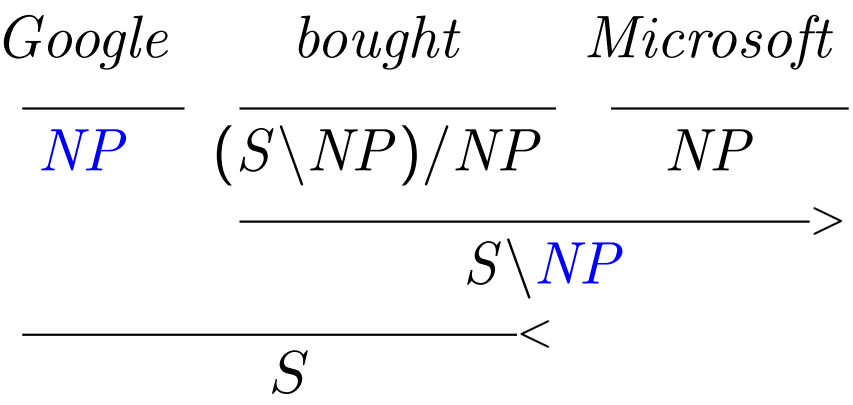
$$\frac{Google}{NP} \quad \frac{bought}{(S\backslash NP)/NP} \quad \frac{Microsoft}{NP}$$

A CCG Derivation

$$\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{\dfrac{}{S\backslash NP}>}$$

$>$ forward application

A CCG Derivation



$>$ forward application

$<$ backward application

Derivation for an Object Extraction

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

$>\mathbf{T}$ type-raising

Derivation for an Object Extraction

The : NP/N | *company* : N | *which* : $(NP\backslash NP)/(S/NP)$ | *Microsoft* : NP | *bought* : $(S\backslash NP)/NP$

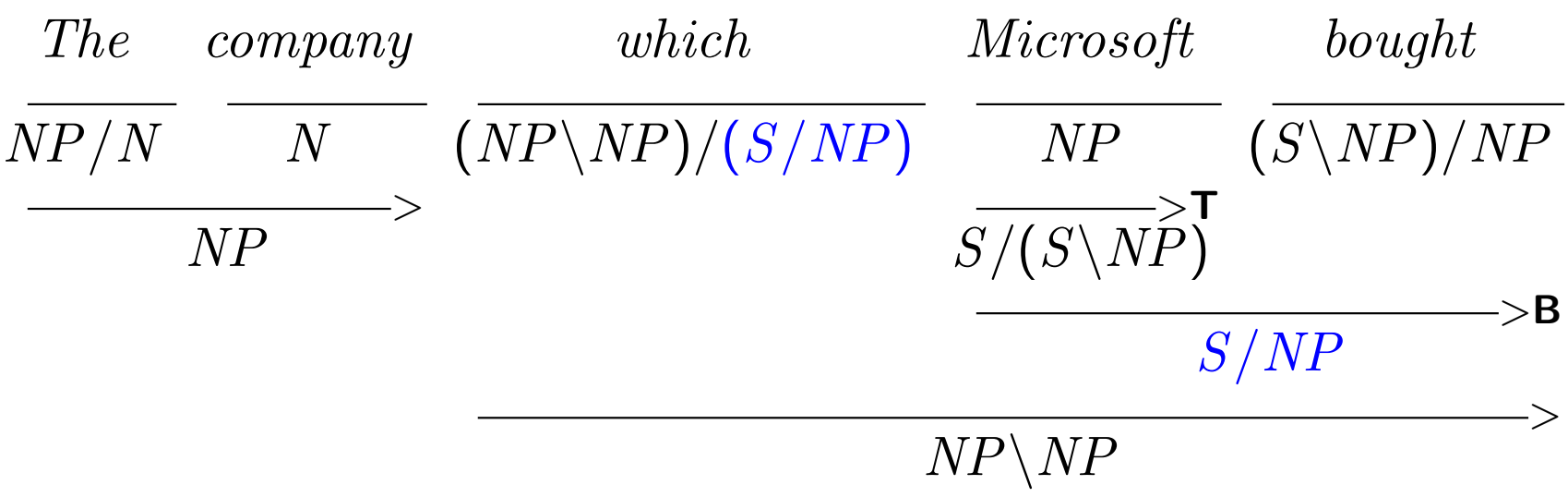
$$NP/N \quad N \;\Rightarrow_{>}\; NP$$

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

$$S/(S\backslash NP) \quad (S\backslash NP)/NP \;\Rightarrow_{>\mathbf{B}}\; S/NP$$

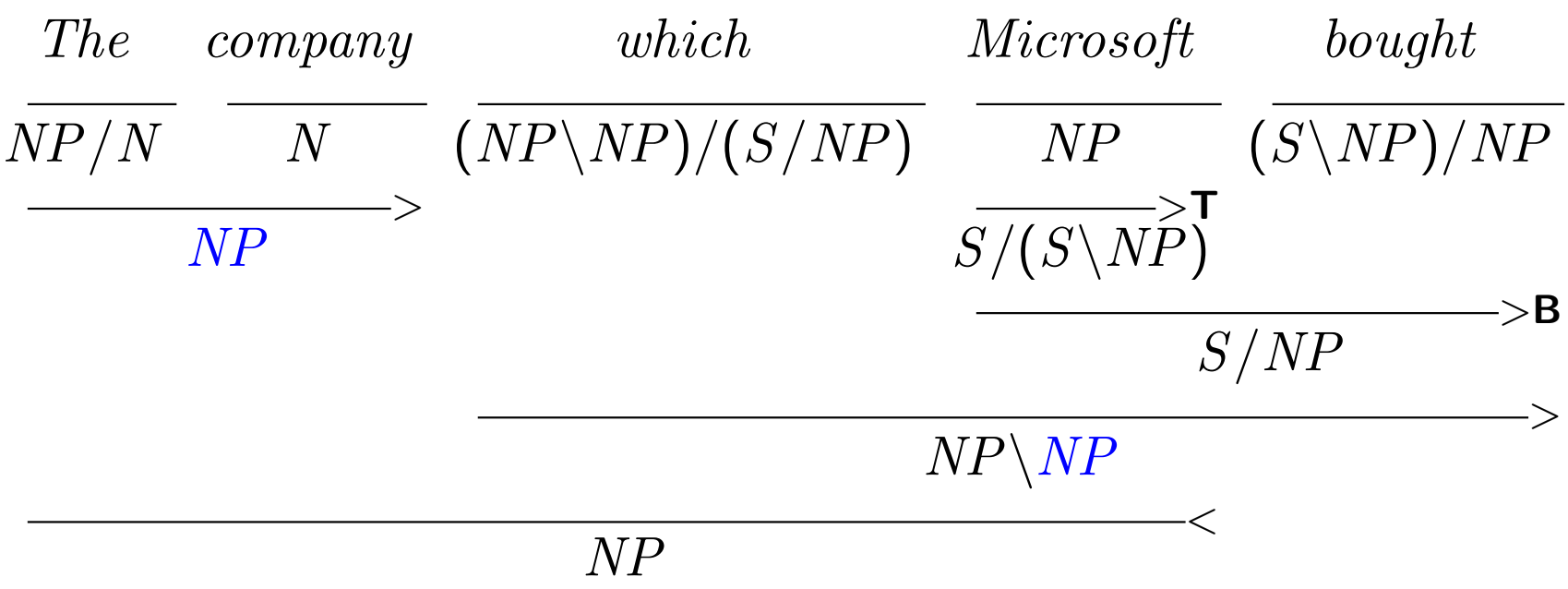
$>\mathbf{T}$ type-raising
$>\mathbf{B}$ forward composition

Derivation for an Object Extraction



- > **T** type-raising
- > **B** forward composition

Derivation for an Object Extraction



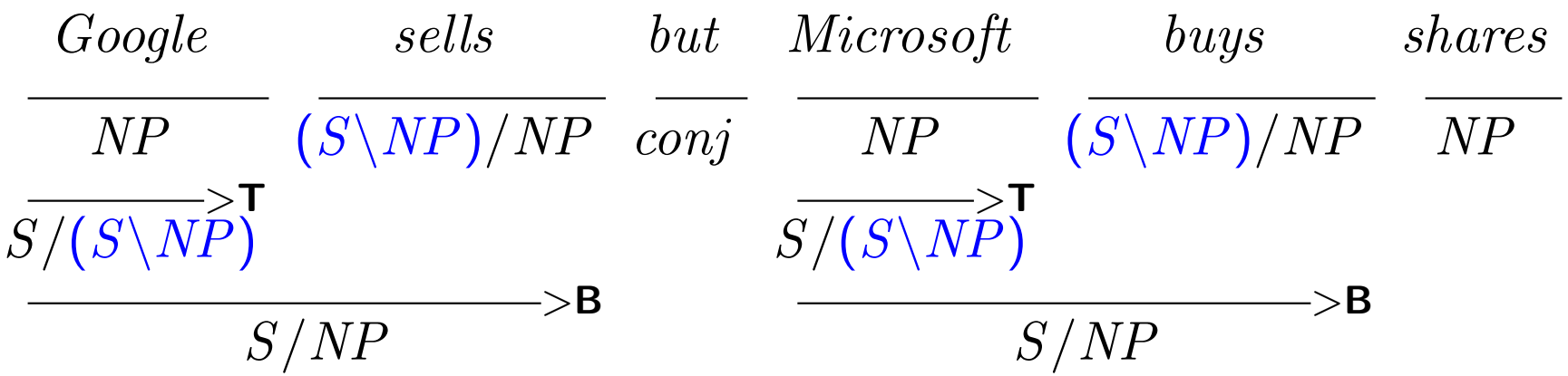
- > **T** type-raising
- > **B** forward composition

"Non-constituents" in CCG

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

$>\mathbf{T}$ type-raising

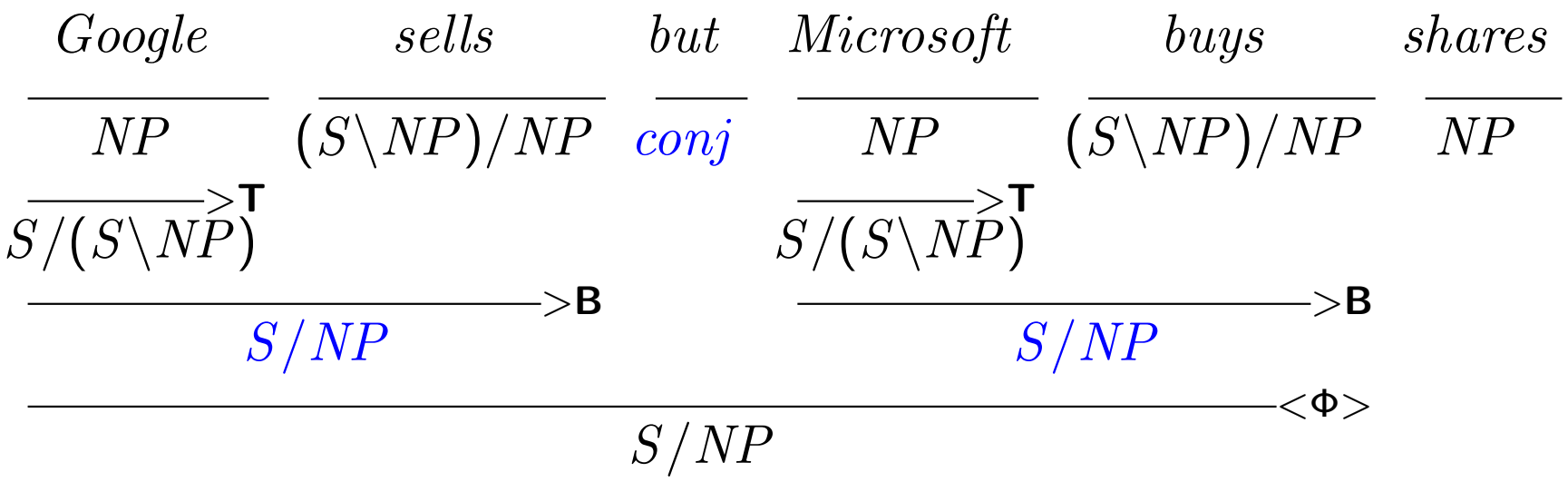
"Non-constituents" in CCG



> **T** type-raising

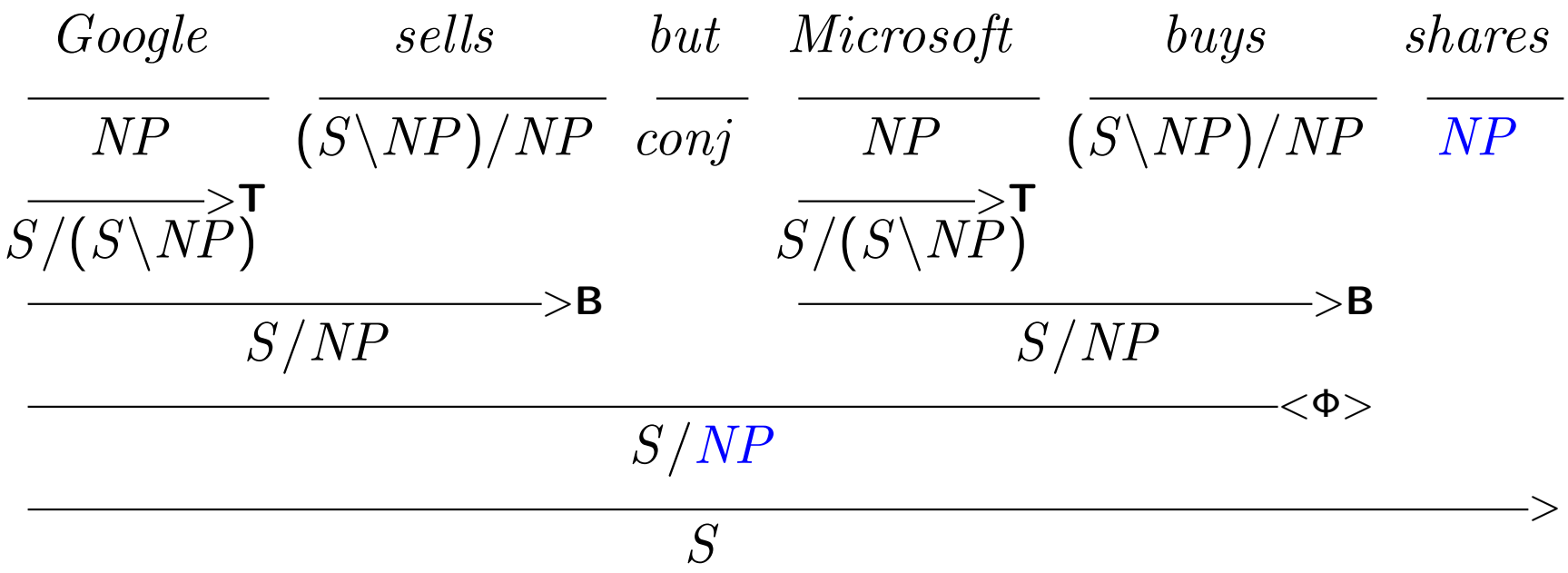
> **B** forward composition

"Non-constituents" in CCG



- > **T** type-raising
- > **B** forward composition

"Non-constituents" in CCG



> **T** type-raising

> **B** forward composition

Derived Structure

$$\dfrac{\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{S\backslash NP}>}{S}<$$

subj(*bought*, *Google*)
obj(*bought*, *Microsoft*)

A Non-Standard CCG Derivation

$$\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{\dfrac{}{S/(S\backslash NP)}\,{>}\mathbf{T}}$$

$>$ **T** type-raising

A Non-Standard CCG Derivation

$$\dfrac{\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{S/(S\backslash NP)}\,{>}\mathbf{T}}{S/NP}\,{>}\mathbf{B}$$

- $>$ **T** type-raising
- $>$ **B** forward composition

A Non-Standard CCG Derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{S/(S\backslash NP)}\,{>}\mathbf{T}}{S/NP}\,{>}\mathbf{B}}{S}\,{>}$$

- $>$ **T** type-raising
- $>$ **B** forward composition

"Spurious" Ambiguity

$$\dfrac{\dfrac{\dfrac{\dfrac{Google}{NP} \quad \dfrac{bought}{(S\backslash NP)/NP} \quad \dfrac{Microsoft}{NP}}{S/(S\backslash NP)}\,{>}\mathbf{T}}{S/NP}\,{>}\mathbf{B}}{S}\,{>}$$

$>$ **T** type-raising

$>$ **B** forward composition

subj(*bought*, *Google*)
obj(*bought*, *Microsoft*)

Spurious Ambiguity

- The existence of spurious ambiguity led many in the NLP community to believe that you couldn't parse efficiently with CCG
- Statistical optimisations allow highly efficent parsing with CCG

CCG Predicate-Argument Dependencies

- Predicate-argument dependencies represented as 4-tuples
- e.g. *company* as the object of *bought* (*IBM bought the company*):

$$\langle \textit{bought}, (S\backslash NP_1)/NP_2, 2, \textit{company} \rangle$$

Parsing with CCG

- Stage 1
  - Assign lexical categories to words in the sentence
  - Use a *supertagger* to assign the categories
    – based on standard Maximum Entropy tagging techniques

Parsing with CCG

- Stage 1
  - Assign lexical categories to words in the sentence
  - Use a *supertagger* to assign the 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

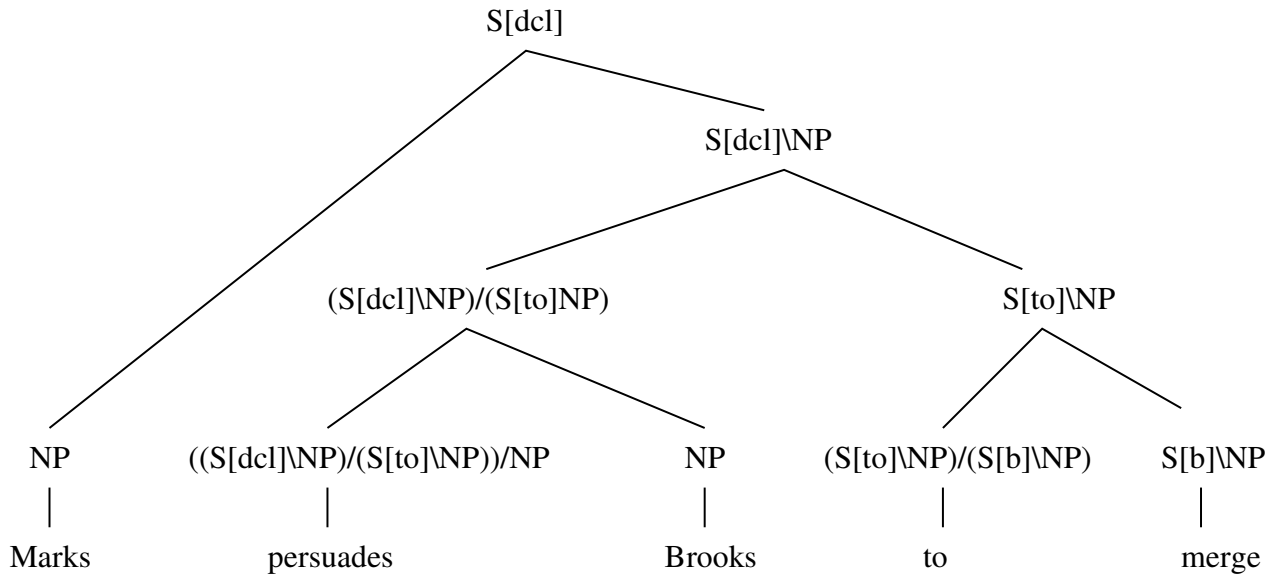
Parsing with CCG

- Stage 1
  - Assign lexical categories to words in the sentence
  - Use a *supertagger* to assign the 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 (*decoding*)
  - Viterbi algorithm finds this efficently

A CCG Treebank: CCGbank

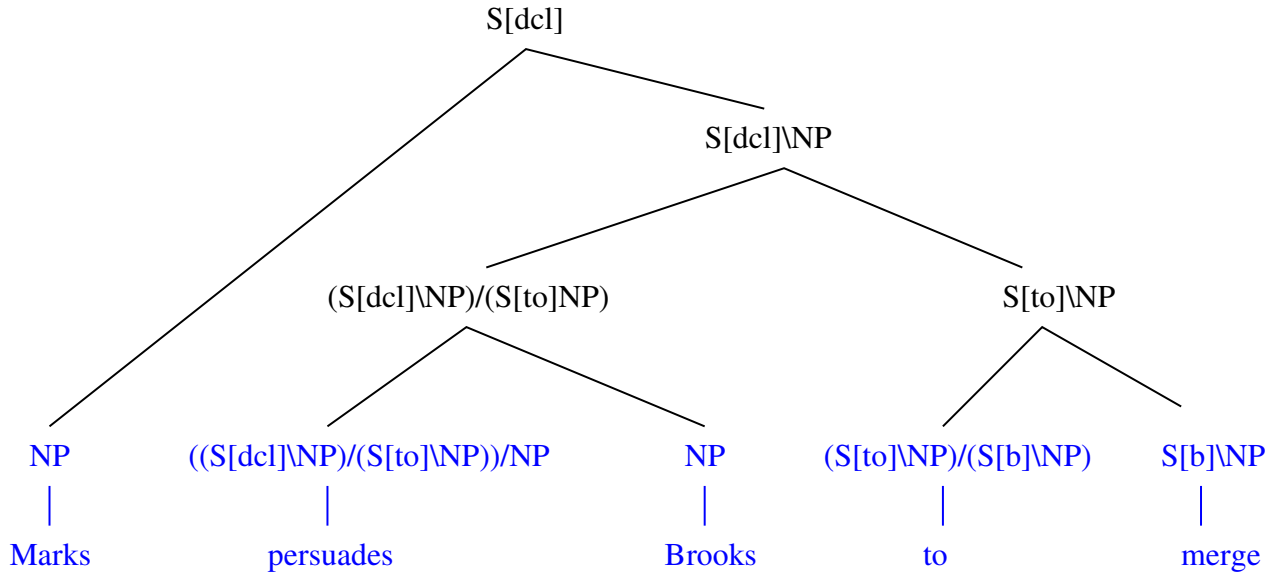
- CCGbank developed by Hockenmaier and Steedman (Hockenmaier, 2003)
- Phrase-structure trees in Penn Treebank (semi-)automatically converted into CCG *normal-form* derivations
- A *normal-form* derivation only uses type-raising and composition when necessary (Eisner, 1996)
  – one way of dealing with spurious ambiguity
- But note phrase-structure trees not isomorphic to CCG analyses (e.g. coordination)

A Treebank of CCG Normal-Form Derivations



- Derivation has been inverted to give a binary tree

Inducing a Grammar



- CCG is a lexicalised grammar, and so the grammar essentially is the lexicon
  – plus a small number of manually defined combinatory rules
- Lexicon can be read off the leaves of the trees

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

Lexical Category Sequence for Newspaper Sentence

$$\underset{(S/S)/NP}{In}\ \underset{NP[nb]/N}{an}\ \underset{N/N}{Oct.}\ \underset{N/N}{19}\ \underset{N}{review}\ \underset{(NP\backslash NP)/NP}{of}\ \underset{NP[nb]/N}{The}$$

$$\underset{N}{Misanthrope}\ \underset{(NP\backslash NP)/NP}{at}\ \underset{N}{Chicago}\ \underset{(NP[nb]/N)\backslash NP}{'s}\ \underset{N/N}{Goodman}\ \underset{N}{Theatre}\ \underset{(NP\backslash NP)/S[dcl]}{-LRB-}$$

$$\underset{N/N}{Revitalized}\ \underset{N}{Classics}\ \underset{(S[dcl]\backslash NP)/NP}{Take}\ \underset{NP[nb]/N}{the}\ \underset{N}{Stage}\ \underset{((S\backslash NP)\backslash(S\backslash NP))/NP}{in}\ \underset{N/N}{Windy}$$

$$\underset{N}{City}\ ,\ \underset{(S\backslash S)/(S\backslash S)}{Leisure}\ \underset{(S\backslash S)/(S\backslash S)}{\&}\ \underset{S\backslash S}{Arts}\ \underset{RRB}{-RRB-}\ ,$$

$$\underset{NP[nb]/N}{the}\ \underset{N}{role}\ \underset{(NP\backslash NP)/NP}{of}\ \underset{N}{Celimene}\ ,\ \underset{S[pss]\backslash NP}{played}\ \underset{((S\backslash NP)\backslash(S\backslash NP))/NP}{by}$$

$$\underset{N/N}{Kim}\ \underset{N}{Cattrall}\ ,\ \underset{(S[dcl]\backslash NP)/(S[pss]\backslash NP)}{was}\ \underset{(S\backslash NP)/(S\backslash NP)}{mistakenly}\ \underset{(S[pss]\backslash NP)/PP}{attributed}\ \underset{PP/NP}{to}$$

$$\underset{N/N}{Christina}\ \underset{N}{Haag}\ .$$

CCG Supertagging

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

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

- Baseline tagging accuracy is $\approx$ 72%

A Maximum Entropy Supertagger

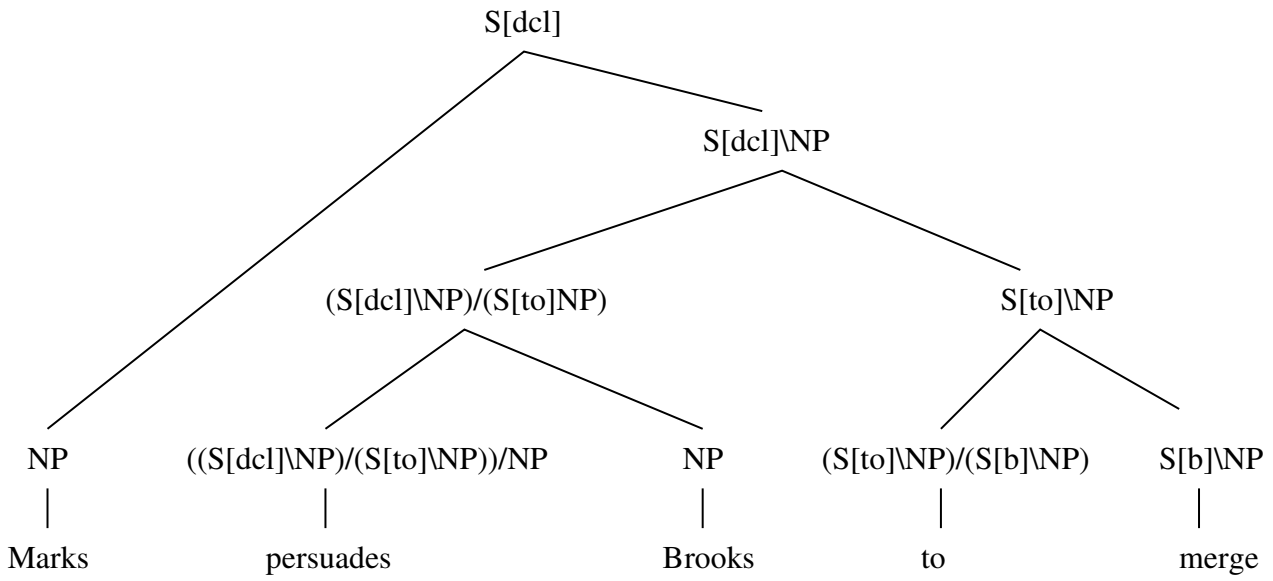
- Maximum Entropy tagging method can be applied to CCG supertagging
- Features are the words and POS tags in a 5-word window, plus the two previously assigned categories
- Per-word tagging accuracy is $\approx$ 92%
- Multi-tagger can assign categories with a per-word accuracy of 97.7% at only 1.4 categories per word

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: $\text{subj}(\text{bought}, \text{IBM})$ (plus distance)
  - "Backing-off" features with words replaced by POS tags
- Use Perceptron training to set the weights

Training Data from CCGbank



$\langle$persuades, $((S[dcl]\backslash NP_1)/(S[to]\backslash NP)_2)/NP_3$, 1, Marks$\rangle$
$\langle$persuades, $((S[dcl]\backslash NP_1)/(S[to]\backslash NP)_2)/NP_3$, 3, Brooks$\rangle$
$\langle$persuades, $((S[dcl]\backslash NP_1)/(S[to]\backslash NP)_2)/NP_3$, 2, merge$\rangle$
$\langle$merge, $S[b]\backslash NP_1$, 1, Brooks$\rangle$

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 $t = 1..T$, $i = 1..N$

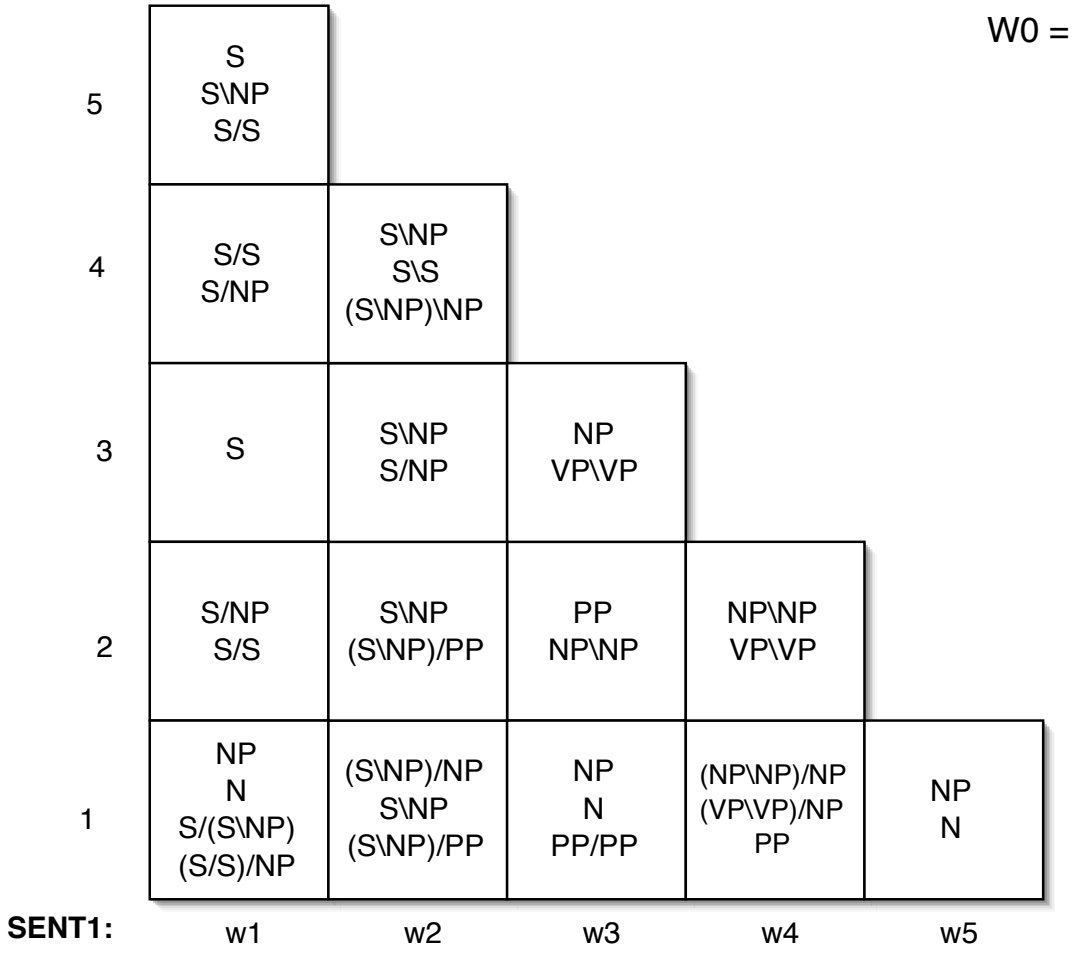
calculate $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \overline{\lambda}$

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>

Perceptron Training

DECODE:

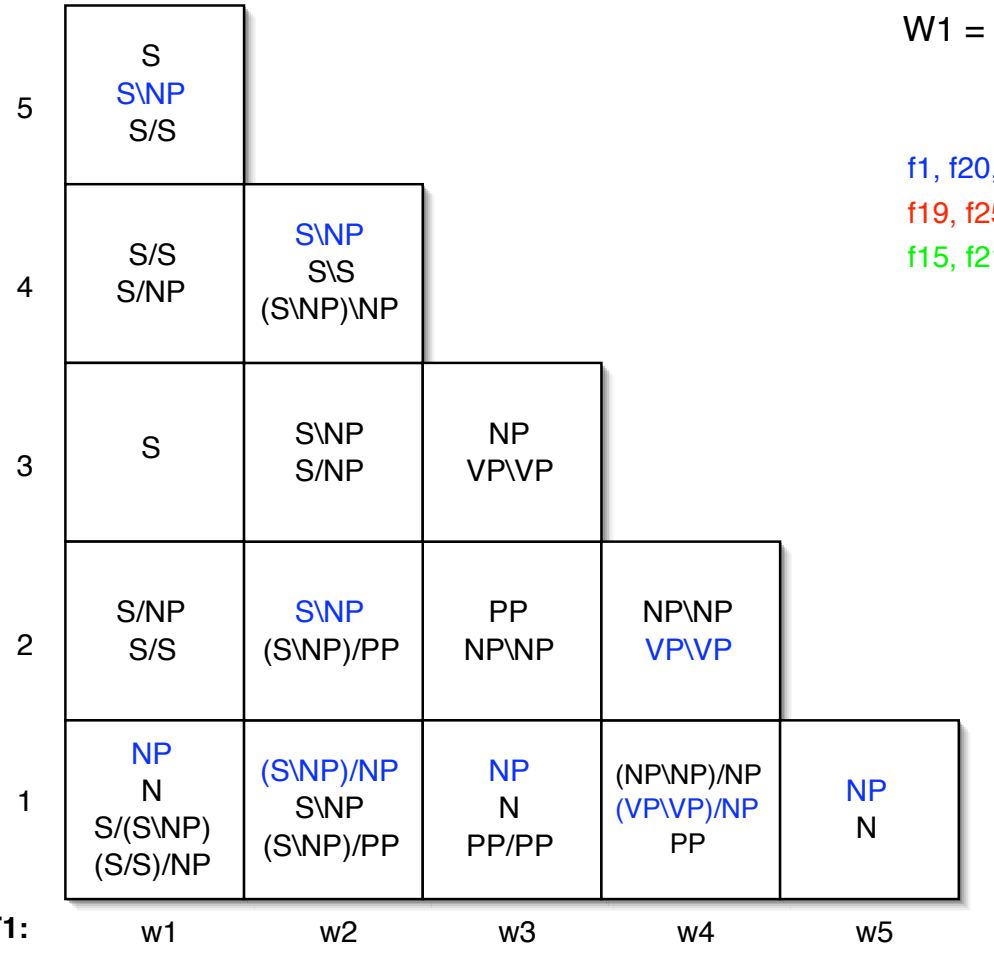
5	S S\NP S/S				
4	S/S S/NP	S\NP S\S (S\NP)\NP			
3	S	S\NP S/NP	NP VP\VP		
2	S/NP S/S	S\NP (S\NP)/PP	PP NP\NP	NP\NP VP\VP	
1	NP N S/(S\NP) (S/S)/NP	(S\NP)/NP S\NP (S\NP)/PP	NP N PP/PP	(NP\NP)/NP (VP\VP)/NP PP	NP N
SENT1:	w1	w2	w3	w4	w5

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:



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

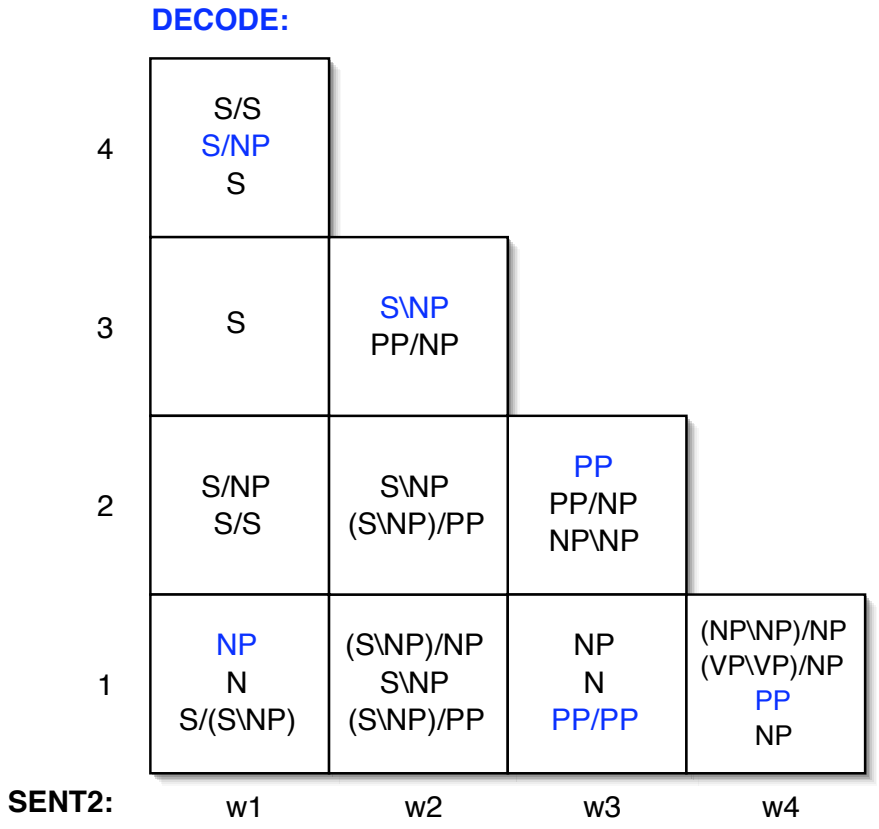
Perceptron Training

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>



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>

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

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

Averaged 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: $\overline{\lambda}$

$$\gamma_s = \sum_{t=1...T, i=1...N} \lambda_s^{t,i} / NT$$

Why is Perceptron Training Hard in Practice?

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

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

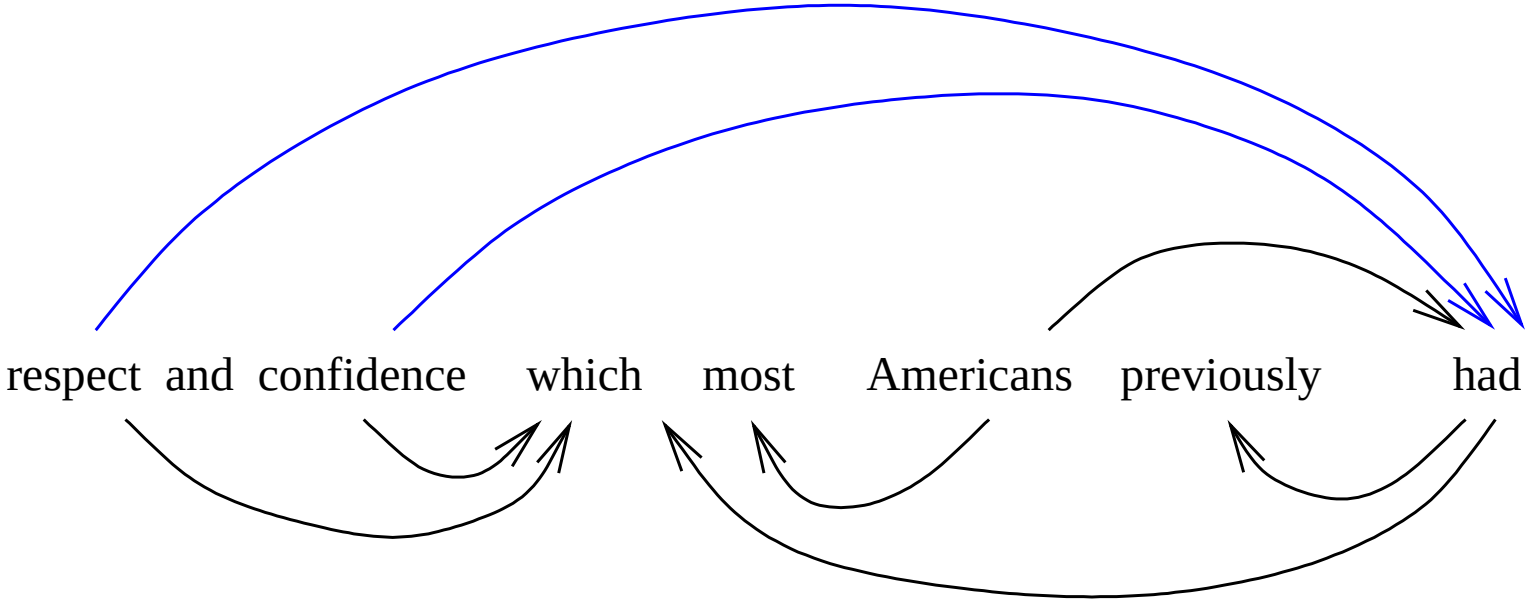
Training in Practice

model	RAM	iterations	time (mins)
perceptron	20 MB	10	312
log-linear (CRF)	19 GB	475	91

Perceptron Training - Theory

- If the training data is "linearly separable", the Perceptron training algorithm will find the weights which separate the data (in a finite number of iterations)
- If the data is "close" to being separable, the Perceptron will make a small number of mistakes
- A small number of mistakes on the training data does imply good performance on unseen data (Collins, 2002)

Parser Output



Parser Output

lexical_item	category	slot	head_of_arg
which	$(NP\backslash NP_1)/(S[dcl]_2/NP)$	2	*had*
which	$(NP\backslash NP_1)/(S[dcl]_2/NP)$	1	*confidence*
which	$(NP\backslash NP_1)/(S[dcl]_2/NP)$	1	*respect*
had	$(S[dcl]\backslash NP_1)/NP_2)$	2	*confidence*
had	$(S[dcl]\backslash NP_1)/NP_2)$	2	*respect*

- Evaluation is in terms of precision and recall of these dependencies
- Note that lexical category has to be correct for dependency to be correct
- Note that both extracted objects have to be recalled in this example to get 100%

Accuracy on Test Data (Section 23 of CCGbank)

model	LP	LR	F	CAT
log-linear (CRF)	87.72	86.32	87.01	94.11
perceptron	87.80	86.46	87.13	94.17

Parser Times for Section 23 (2,407 sentences)

Parser	time (mins)
Collins	45
Charniak	28
Sagae	11
CCG	1.2

- High speed comes from the supertagger (and highly engineered C++)

Formalism Independent Evaluation

- Accuracy figures on CCGbank are inflated and not comparable with non-CCG parser scores
- Need a formalism-independent resource whose output the CCG parser can produce
- Important problem for current parsing research
- DepBank is an ideal candidate

DepBank

- 700 sentences of Section 23 manually annotated with Grammatical Relations (GRs) by Briscoe and Carroll
- So just need to translate CCG dependencies into DepBank GRs
- Sounds easy?
- Mapping into DepBank is the best method we currently have for comparing parsers across formalisms

Grammatical Relations

conj	coordinator
aux	auxiliary
det	determiner
ncmod	non-clausal modifier
xmod	unsaturated predicative modifier
cmod	saturated clausal modifier
pmod	PP modifier with a PP complement
ncsubj	non-clausal subject
xsubj	unsaturated predicative subject
csubj	saturated clausal subject
dobj	direct object
obj2	second object
iobj	indirect object
pcomp	PP which is a PP complement
xcomp	unsaturated VP complement
ccomp	saturated clausal complement
ta	textual adjunct delimited by punctuation

Example CCG Dependencies and GRs

But Mr. Barnum called that a worst-case scenario.

```
Mr._2 N/N_1 1 Barnum_3
called_4 ((S[dcl]\NP_1)/NP_2)/NP_3 3 that_5
worst-case_7 N/N_1 1 scenario_8
a_6 NP[nb]/N_1 1 scenario_8
called_4 ((S[dcl]\NP_1)/NP_2)/NP_3 2 scenario_8
called_4 ((S[dcl]\NP_1)/NP_2)/NP_3 1 Barnum_3
But_1 S[X]/S[X]_1 1 called_4

(ncmod _ Barnum_3 Mr._2)
(obj2 called_4 that_5)
(ncmod _ scenario_8 worst-case_7)
(det scenario_8 a_6)
(dobj called_4 scenario_8)
(ncsubj called_4 Barnum_3 _)
(conj called_4 But_1)
```

Mapping CCG Dependencies to GRs

CCG lexical category	arg slot	GR
$(S[dcl]\backslash NP_1)/NP_2$	1	(ncsubj %l %f _)
$(S[dcl]\backslash NP_1)/NP_2$	2	(dobj %l %f)
$(S\backslash NP)/(S\backslash NP)_1$	1	(ncmod _ %f %l)
$(NP\backslash NP_1)/NP_2$	1	(ncmod _ %f %l)
$(NP\backslash NP_1)/NP_2$	2	(dobj %l %f)
$NP[nb]/N_1$	1	(det %f %l)
$(NP\backslash NP_1)/(S[pss]\backslash NP)_2$	1	(xmod _ %f %l)
$(NP\backslash NP_1)/(S[pss]\backslash NP)_2$	2	(xcomp _ %l %f)
$((S\backslash NP)\backslash(S\backslash NP)_1)/S[dcl]_2$	1	(cmod _ %f %l)
$((S\backslash NP)\backslash(S\backslash NP)_1)/S[dcl]_2$	2	(ccomp _ %l %f)
$((S[dcl]\backslash NP_1)/NP_2)/NP_3$	2	(obj2 %l %f)
$(S[dcl]\backslash NP_1)/(S[b]\backslash NP)_2$	2	(aux %f %l)

Mapping is Many-to-Many

CCG lexical category	slot	GR	constraint
$(S[dcl]\backslash NP_1)/NP_2$	2	`(xcomp _ %l %f)`	word=be
		`(dobj %l %f)`	
$(S[dcl]\backslash NP_1)/PP_2$	2	`(iobj %l %f)`	cat=PP/NP
		`(xcomp _ %l %f)`	cat=$PP/(S[ng]\backslash NP)$

The parent *is Imperial*

The parent *sold Imperial*

The loss *stems from* several factors

The future *depends on* building ties

Other Difficulties

- Different "heads":
  The group said it would consider withholding royalty payments Sen. Mitchell, who had proposed the streamlining
- More subjects in CCGbank
- Coordinations as arguments:
  The president and chief executive officer said the loss stems from several factors.
- Argument/adjunct distinction (`ncmod` vs. `iobj`)
- Trivial differences such as tokenisation, shorter sentences in DepBank(!)

Oracle Experiment

- Take the *gold-standard* CCGbank dependencies from Section 23 and transform into GRs
- Provides some indication of the effectiveness of the mapping
- Comparing with DepBank gives the score a perfect CCGbank parser would achieve
  (so provides an upper bound for the CCG parser)
- Other researchers have not done this

Oracle Experiment

- Take the *gold-standard* CCGbank dependencies from Section 23 and transform into GRs
- Provides some indication of the effectiveness of the mapping
- Comparing with DepBank gives the score a perfect CCGbank parser would achieve
  (so provides an upper bound for the CCG parser)
- Other researchers have not done this
- F-score of 84.8%

Comparison with RASP

CCG	RASP
Domain-dependent (WSJ)	Domain-independent
Automatically-extracted grammar	Manually created grammar
Lexicalised parsing model	Unlexicalised model
Derived from the Penn Treebank	Trained on Susanne

RASP provides a strong baseline on DepBank

Final Results*

	P	R	F
RASP	77.66	74.98	76.29
CCG	82.39	81.17	81.77
CCGbank	86.86	82.75	84.76

*thanks to Rebecca Watson and Ted Briscoe for help with the evaluation

Conclusion*

- Combination of Machine Learning methods and linguistically motivated grammars is bearing fruit
  – we now have robust, efficient and accurate parsers of real text producing meaningful output
- CCG parser is accurate and efficient
  – we have parsed the entire Gigaword corpus in less than 5 days using only 18 machines
- Research questions:
  - Porting to biomedical domain
  - More sophisticated online learning algorithms
  - More complex feature sets
  - Semi-supervised learning

*Parser, including source code, is freely available

Towards Richer Output (Bos, 2005)

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