ULL'13

Lecture 9: Taking Statistical Learning to its Limits, and Beyond

Jelle Zuidema

Learning a grammar

- Choose a generative model
  - HMM, PCFG, PTSG, PTAG, …
- Choose an objective function
  - Maximum Likelihood, Bayesian …
- Choose an optimization strategy
  - Stochastic hillclimbing
- Choose a dataset
- Find the generative model that maximizes the objective function on the dataset!

- How can we learn the building blocks and rules of combination on natural language from (unlabeled) data?

- Formal languages: form-meaning associations, finite-state machines, context-free grammars, tree substitution grammars, dependency grammars, lambda calculus

- Statistical Models: Markov models, HMMs, PCFGs, PPBDG

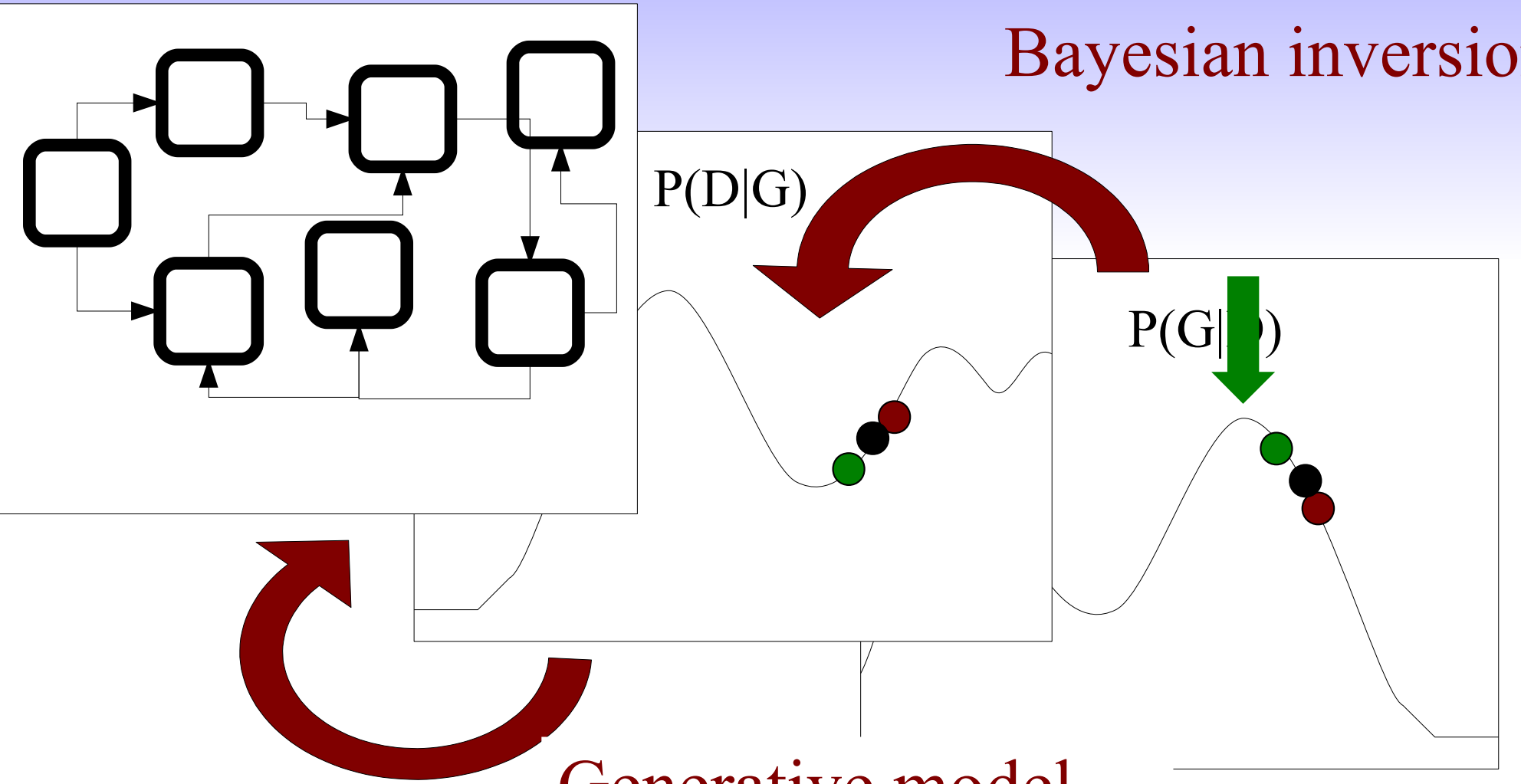
- Statistical Inference: EM, Maximum Likelihood, Sampling methods, Posterior Mode

1. Taking statistical learning of grammars from unlabeled data to its limits: Cohn et al. 2010

2. A step back: Iterated Learning: Zuidema 2003

3. Algorithmicly defined learning procedures: Seginer 2007

MAP



Cohn et al. 2010

All tricks from the book:

- Probabilistic Tree Substitution Grammars (like DOP, "commitment")
- Pitman-Yor Chinese Restaurant Processes to define prior over PTSGs (and likelihood of the corpus).
- DMV (like Klein & Manning, 2004)
- Split-head encoding (like Johnson)

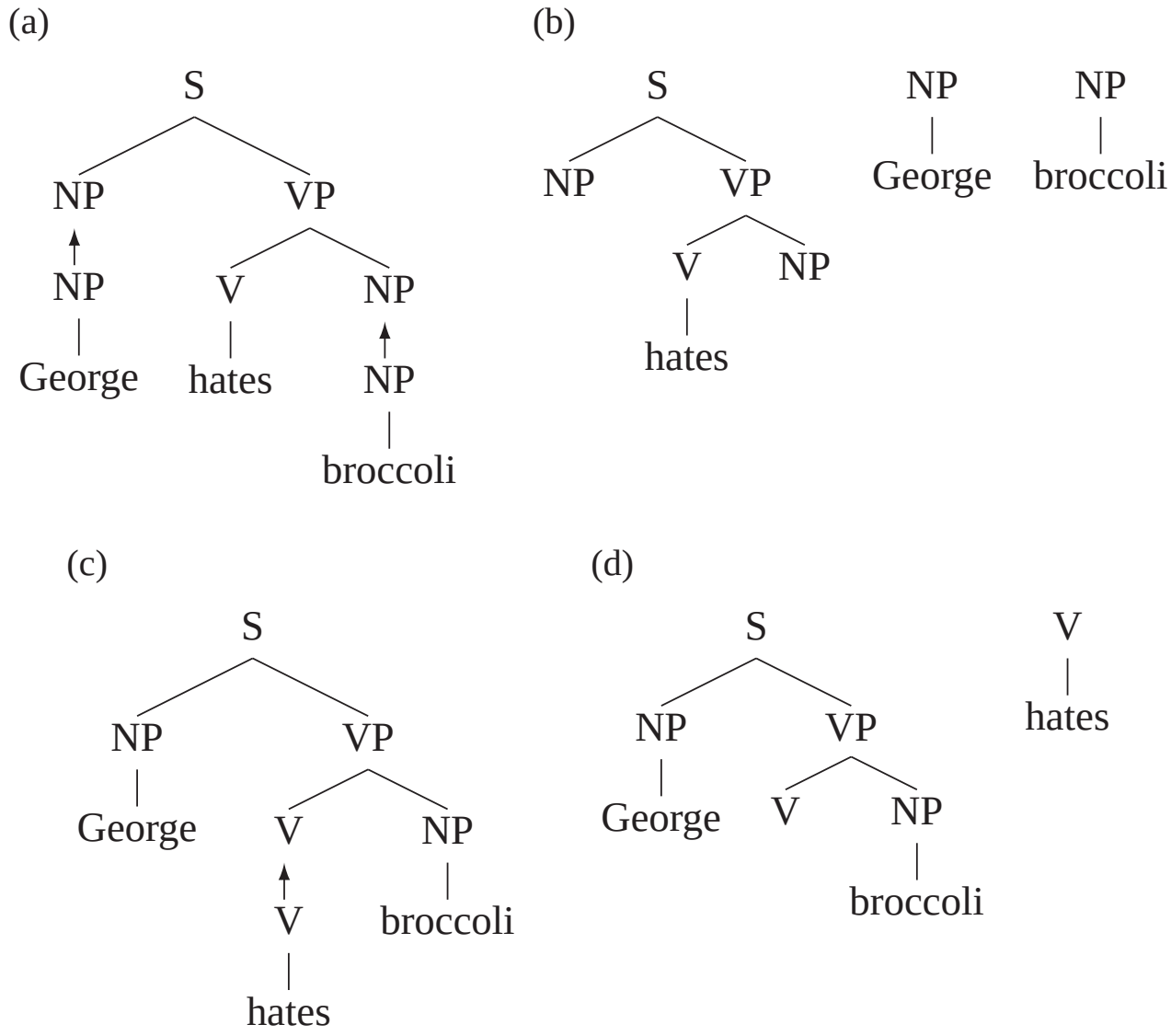


Figure 1: Example derivations for the same tree, where arrows indicate substitution sites. The left figures (a) and (c) show two different derivations and the right figures (b) and (d) show the elementary trees used in the respective derivation.

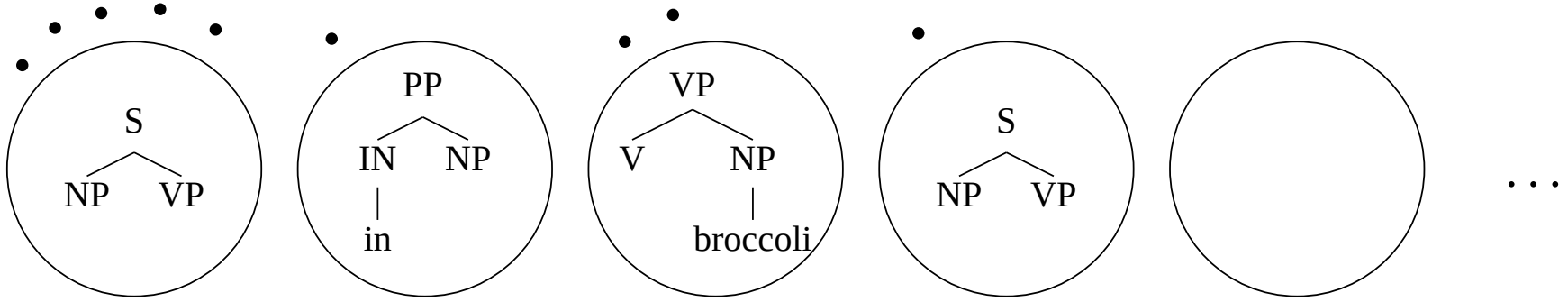


Figure 4: The Pitman-Yor process, illustrated as a labelled Chinese restaurant process. In this example, $\mathbf{z}_{-10} = (1, 2, 1, 1, 3, 1, 1, 4, 3)$ and each table k is labelled with an elementary tree ℓ_k. Black dots indicate the number of occurrences of each tree in $\mathbf{e} = (\ell_1, \ell_2, \ell_1, \ell_1, \ell_3, \ell_1, \ell_1, \ell_4, \ell_3)$. In this illustration, which corresponds to the model given in (2), a single Pitman-Yor process is used to generate all elementary trees, so the trees do not necessarily fit together properly. Our complete model, defined in (5), would have a separate Pitman-Yor restaurant for each root category.

Equation 4 shows that, like other PYP and DP models, this model can be viewed as a *cache model*, where e_i can be generated in one of two ways: by drawing from the base distribution or by drawing from a cache of previously generated elementary trees, where the probability of any particular elementary tree is proportional to the discounted frequency of that tree. This view makes

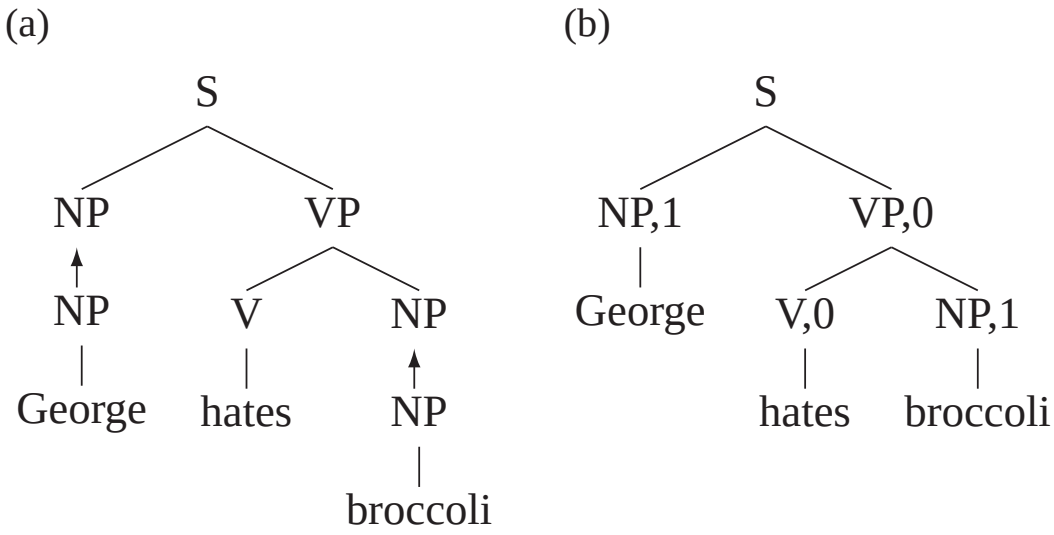


Figure 5: Gibbs sampler state (b) corresponding to the example derivation (a) (reproduced from Figure 1a). Each node is labelled with its substitution variable.

4.1 Local Sampler

Sampling techniques

MCMC / Gibbs sampling:

- for a variable **v** with n components,
- iteratively reestimate v_i for every $i=1...n$ from $P(v_i \mid v_1...v_{i-1},v_{i+1}...v_n)$.

Converges to $P(\mathbf{v})$.

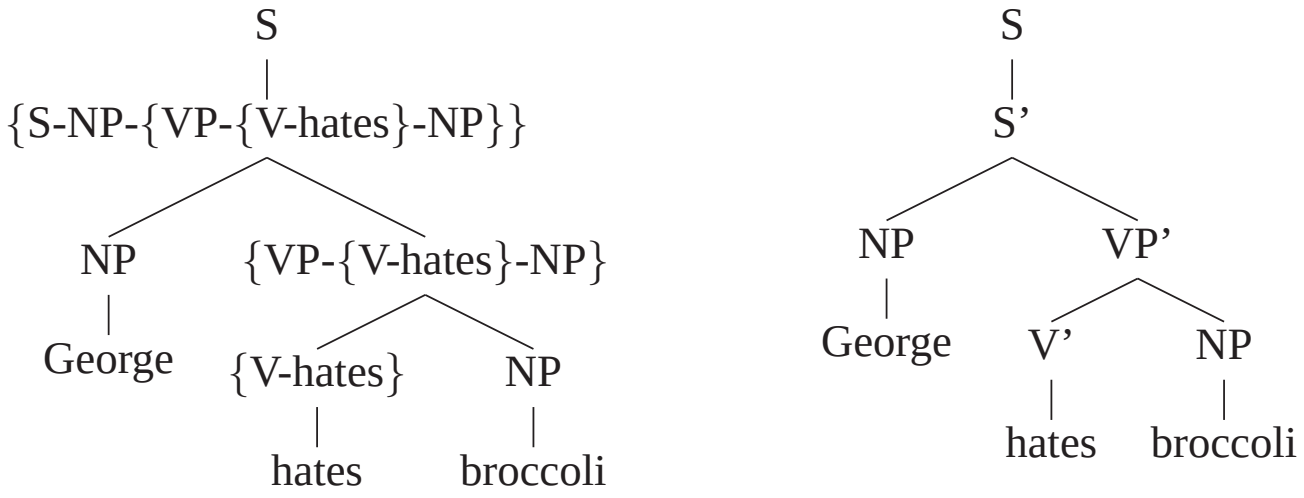


Figure 7: Example trees under the grammar transform, which both encode the same TSG derivation from Figure 1a. The left tree encodes that the S → NP (VP (V hates) NP elementary tree was drawn from the cache, while for the right tree this same elementary tree was drawn from the base distribution (the count and base terms in (11), respectively).

site, $x_d = 1$, otherwise it is an internal node, $x_d = 0$. For example, both trees in Figure 7 encode that both NP nodes are substitution sites and that the VP and V nodes are not substitution sites (the same configuration as Figure 5).

The time complexity of the constrained inside algorithm is linear in the size of the tree and the

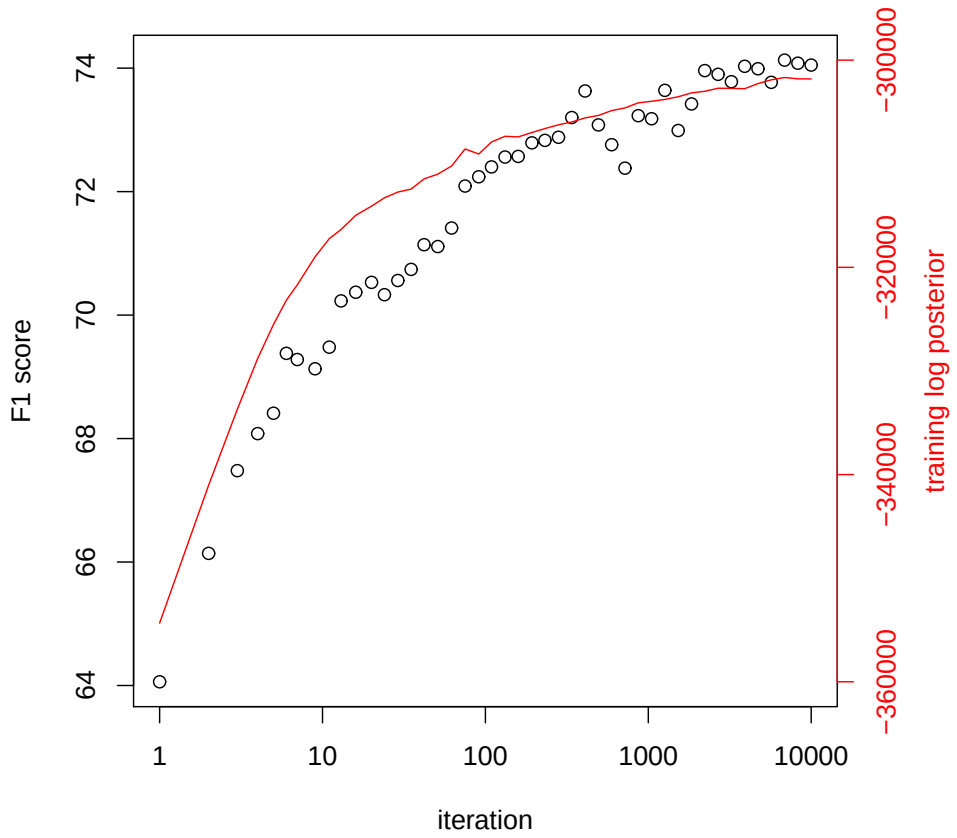


Figure 10: Likelihood and generalisation F1 are highly correlated. The black circles show the

	≤ 40		**all**	
Parser	**F1**	**EX**	**F1**	**EX**
MLE PCFG	64.2	7.2	63.1	6.7
TSG PYP Viterbi	83.6	24.6	82.7	22.9
TSG PYP MPD	84.2	27.2	83.3	25.4
TSG PYP MPT	84.7	28.0	83.8	26.2
TSG PYP MER	85.4	27.2	84.7	25.8
DOP (Zuidema, 2007)			83.8	26.9
Berkeley parser (Petrov and Klein, 2007)	90.6		90.0	
Berkeley parser (restricted)	87.3	31.0	86.6	29.0
Reranking parser (Charniak and Johnson, 2005)	92.0		91.4	

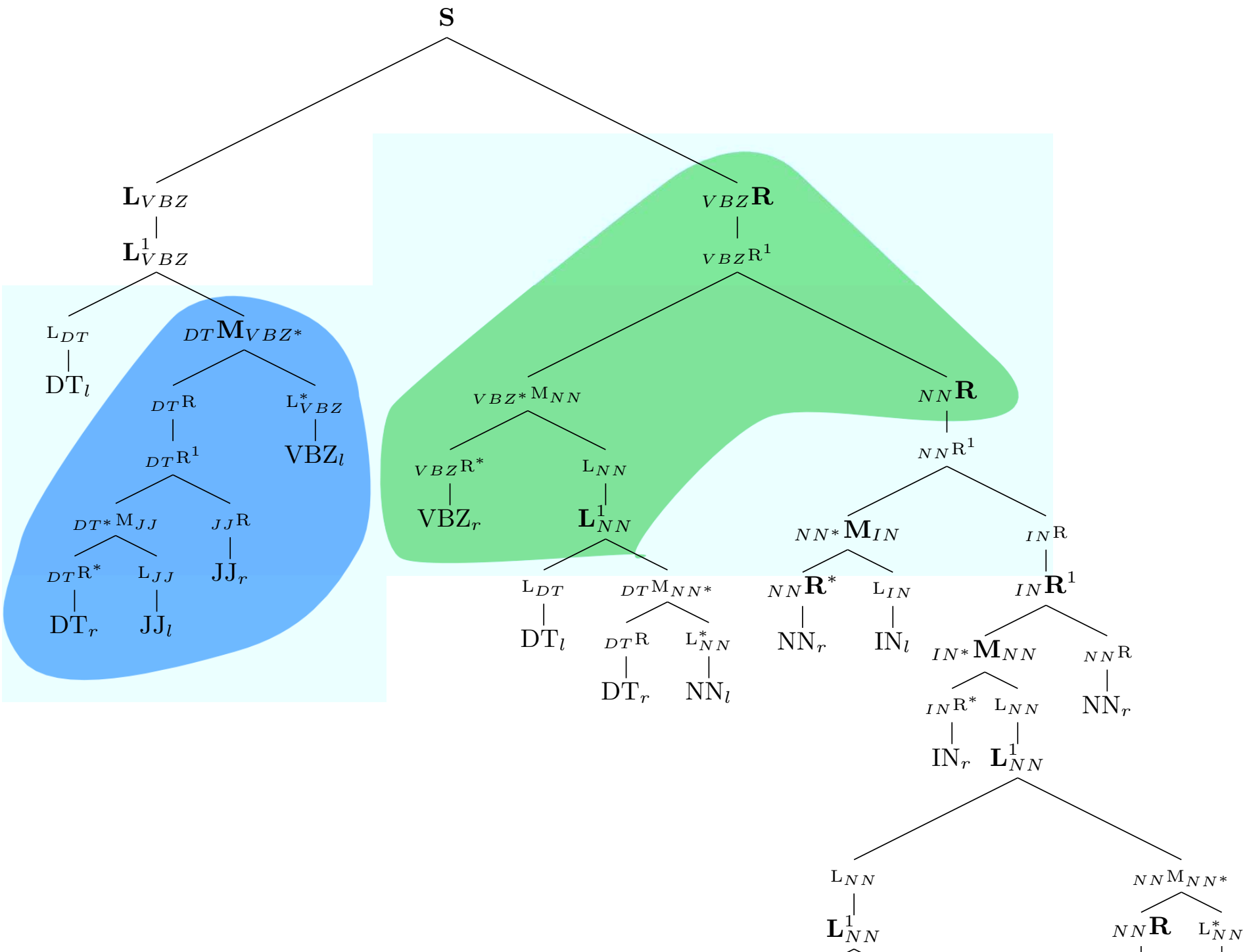
Table 4: Full treebank testing results showing labelled F1 and exact match accuracy for sentences of up to 40 words, and for all sentences. The results of several treebank parsers are also shown (as reported in the literature, hence the missing values), representing a baseline (PCFG), systems similar to our own (DOP, Berkeley) and state-of-the-art (Berkeley, Reranking parser). Berkeley (restricted)

Model	Viterbi	MPD	MPP	MER	# rules
PCFG	60.20	60.20	60.20	-	3500
TSG PYP	74.90	76.68	77.17	78.59	25746
TSG DP	74.70	75.86	76.24	77.91	25339
Berkeley parser ($\tau = 2$)	71.93	71.93	-	74.32	16168
Berkeley parser ($\tau = 5$)	75.33	75.33	-	77.93	39758

Table 3: Development results for models trained on section 2 of the Penn treebank, showing labelled constituent F1 and the grammar size. For the TSG models the grammar size reported is the number of CFG productions in the transformed MAP PCFG approximation. Unknown word models are applied to words occurring less than two times (TSG models and Berkeley $\tau = 2$) or less than five times (Berkeley $\tau = 5$).

the TSG can make a large difference. Surprisingly, the MPP technique is only slightly better than the MPD approach, suggesting that derivational ambiguity is not as much of a problem as previously thought (Bod, 2003). Also surprising is the fact that exact Viterbi parsing under the MAP approximation is much worse than the MPD method which uses an approximate search technique under

(a) TSG-DMV representation. Large bold nodes indicate substitution points.



Model	Initialiser	Directed Attachment Accuracy % on Section 23		
		$\|w\| \leq 10$	$\|w\| \leq 20$	$\|w\| \leq \infty$
Attach-Right	-	38.4	33.4	31.7
EM (Klein and Manning, 2004)	Harmonic	46.1	39.9	35.9
Dirichlet (Cohen et al., 2009)	Harmonic	46.1	40.6	36.9
LN (Cohen et al., 2009)	Harmonic	59.4	45.9	40.5
SLN, TIE V&N (Cohen and Smith, 2009)	Harmonic	61.3	47.4	41.4
DMV (Headden III et al., 2009)	Random	$55.7_{\sigma=8.0}$	-	-
DMV smoothed (Headden III et al., 2009)	Random	$61.2_{\sigma=1.2}$	-	-
EVG smoothed (Headden III et al., 2009)	Random	$65.0_{\sigma=5.7}$	-	-
L-EVG smoothed (Headden III et al., 2009)	Random	$\mathbf{68.8}_{\sigma=4.5}$	-	-
Less is More WSJ15 (Spitkovsky et al., 2010)	Harmonic	56.2	48.2	44.1
Leap Frog WSJ45 (Spitkovsky et al., 2010)	Harmonic	57.1	48.7	45.0
Adaptor Grammar (Cohen et al., 2010)	Harmonic	50.2	-	-
TSG-DMV	Harmonic	$65.9_{\sigma=2.4}$	$58.3_{\sigma=2.3}$	$53.1_{\sigma=2.4}$
TSG-DMV WSJ15	Harmonic	$66.4_{\sigma=1.7}$	$\mathbf{58.5}_{\sigma=1.7}$	$\mathbf{53.4}_{\sigma=1.8}$
Supervised MLE (Cohen and Smith, 2009)	-	84.5	74.9	68.8

A step back: how I got into all this

Identification in the limit

(Gold, 1967)

> *"A class of language will be called* identifiable in the limit *[...] if there is an effective learner [...] with the following property: Given any language of the class and given any allowable training sequence for this language, the language will be identified in the limit."*

- Positive evidence: *Text*
- Negative evidence: *Informant sequence*

Context-free grammars are not learnable from text

Infinite languages can not be identified, because there exists an infinite sequence of finite languages that are indistinguishable for any amount of training samples.. I.e. no matter how many examples you have seen, you'll never know whether you've seen the whole language or whether you should generalize to (infinitely) more.

infinite language	*finite languages*
$S \mapsto Sa$	$S \mapsto a$
$S \mapsto a$	$S \mapsto aa$
	$S \mapsto aaa$
	$S \mapsto aaaa$
	$S \mapsto aaaaa$
	...

Principle & Parameter grammars are learnable from text

(Wexler & Culicover, 1980)

E.g., by *identification through enumeration*. The algorithm considers a finite number of hypotheses; it sticks to an hypothesis until it receives a counter example; it will always receive a counter example within a finite amount of time. If it considers hypotheses in the right order, it will therefore always arrive and stay at the correct hypothesis within a finite amount of time.

$S \mapsto aSb$ $S \mapsto ab$	$S \mapsto aS \mid aA$ $A \mapsto Ab \mid b$
aabb	*aabb*
aaaabbbb	*aaaabbbb*
aaaaaabbbbbb	*aaaaaabbbbbb*
aaaaabbbbb	*aabbbbb*

"The basic results of the field [of learnability theory] include the formal, mathematical demonstration that without serious constraints on the nature of human grammar, no possible learning algorithm can in fact learn the class of human grammars."

(Wexler, 1999, "Innateness of Language", MIT Encyclopedia of Cognitive Science)

Iterated Learning

(Zuidema, 2003; Kirby 1999)

Representation

Context-free grammars

- Rules of the forms: $A \mapsto t$, $A \mapsto BC$, $A \mapsto Bt$
- Start symbol S, terminal symbols (lexicon), non-terminal symbols

Lexical

S ↦ mary shouts
S ↦ angry mary shouts
S ↦ ...

Combinatorial

S ↦ N V
N ↦ mary
N ↦ angry mary
N ↦ ...
V ↦ walks

Recursive

S ↦ N V
N ↦ A N
N ↦ mary
A ↦ beautiful
A ↦ angry
V ↦ walks

Unsupervised learning: learning a grammar from a string set

Incorporation: extend the language, such that it includes the encountered string

Compression: substitute frequent and long substrings with a nonterminal (the grammar becomes smaller and the language remains unchanged)

Generalization: equate two nonterminals if they occur frequently in the same context

Example

Training sentences: abcd, abcababcd, abcabcabcd

(a) Incorporation

$S \mapsto abcd$
$S \mapsto abcabcd$
$S \mapsto abcabcabcd$

(b) Compression

$S \mapsto abcd$
$S \mapsto Xd$
$S \mapsto Xabcd$
$X \mapsto abcabc$

(c) Compression

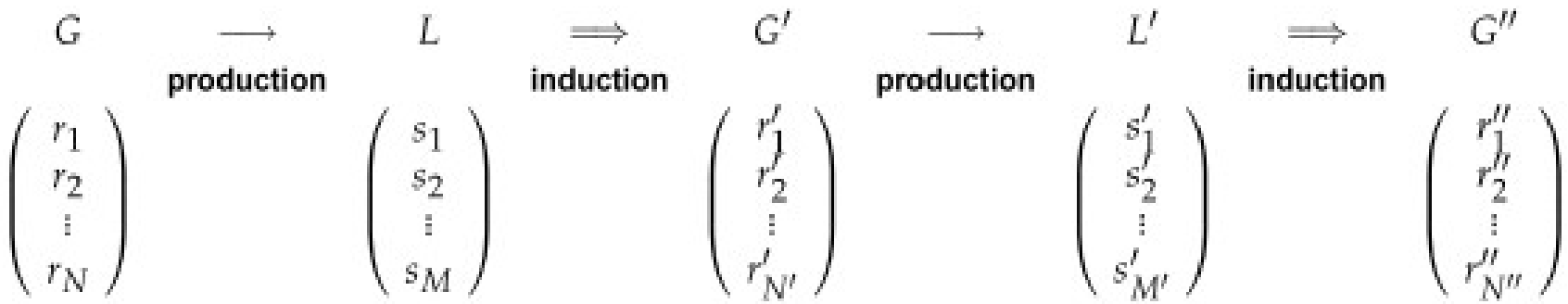
$S \mapsto Yd$
$S \mapsto Xd$
$S \mapsto Xabcd$
$X \mapsto YY$
$Y \mapsto abc$

(d) Generalization

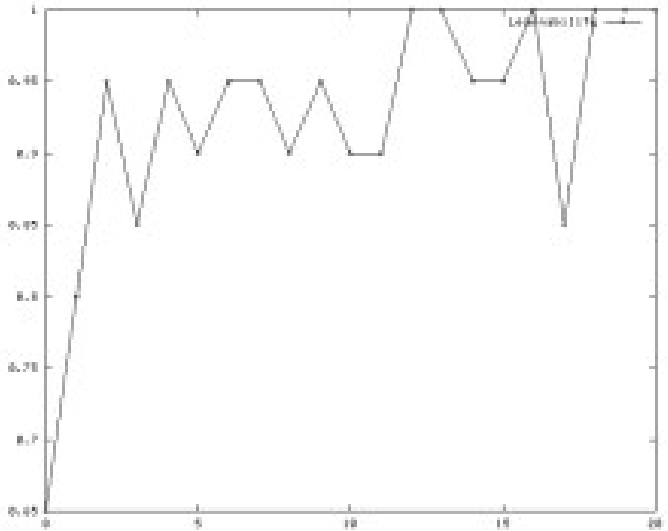
$S \mapsto Xd$
$S \mapsto Xabcd$
$X \mapsto XX$
$X \mapsto abc$

Iterated Learning

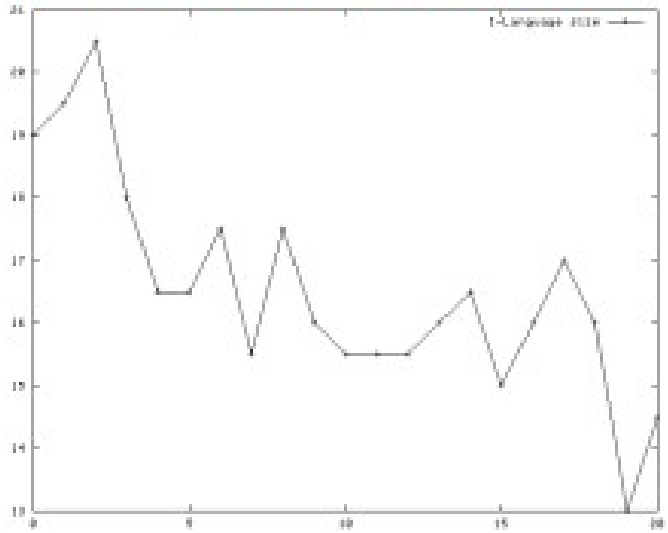
Individuals in a **chain** learn from the previous individual and teach to the next



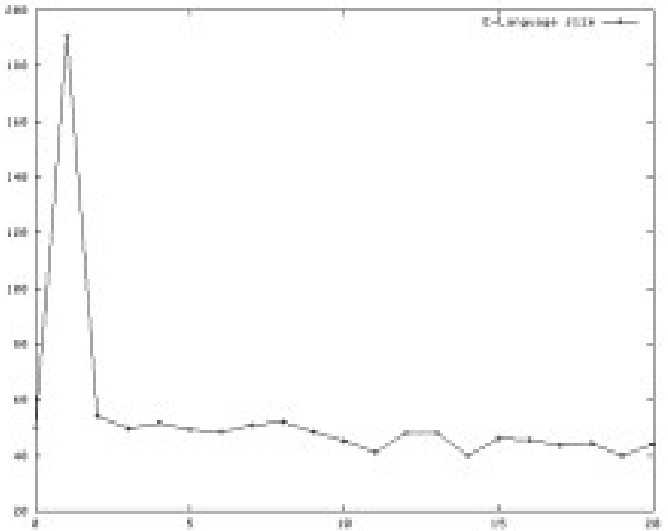
Iterated Learning - results



learnability



number of rules



number of sentences

Parameters: $V_t = \{a, b, c, d\}$, $V_{nt} = \{S, X, Y, Z, A, B, C\}$, T=30, E=20, l_0=3.
Shown are the average values of 2 simulations.

Example

1. “ada”, “ddac”, “adba”, “bcbd”, “cdca”
2. “dcac”, “bcac”, “caac”, “daac”
3. $S \mapsto dcX$, $S \mapsto bcX$, $S \mapsto caX$, $S \mapsto daX$, and $X \mapsto ac$
4. $Y \mapsto b$
5. “dcac”, “bcac”, “caac”, “bcb”, “cab”, “dab”,

Cultural Adaptation

1. Languages are transmitted culturally and are subject to change

2. Languages will change more if they are difficult to learn

3. Over time, languages that are easy to learn are more likely to occur

$\longrightarrow$ "poverty of stimulus" arguments that postulate extensive innate "knowledge of language" in addition to a "language acquisition device" loose much of their force.

If the languages we will ever have to learn (i.e. the set of possible targets) are shaped by the learning procedure, perhaps it makes sense to try to focus on finding elegant/simple learning procedure rather than elegant/simple sets of formal languages.

Miniprojects

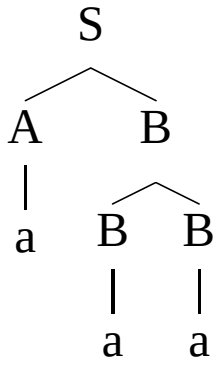
Parsing trees

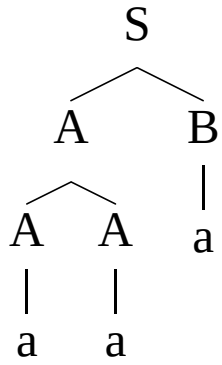
Willem Zuidema

February 28, 2011

Consider the following PCFG and two parses of the string 'aaa':

S → A B	1.0
A → A A	0.3
A → a	0.7
B → B B	0.8
B → a	0.2





The left tree has probability $.7 \times .8 \times .2 \times .2 = .0224$, while the right tree has probability $.3 \times .7 \times .7 \times .2 = .0294$. In a standard PCFG-parser (e.g., BitPar), we can easily get those parses of 'aaa' and their probabilities. However, if we only want to know the probability of a given parse tree, we are waisting computing time by calculating all parse trees and their probabilities. We like 'trick' an off-the-shelf parser into computing the parse probability of only a specific parse.

The approach is based on defininining a transform of the original grammar, and feeding the parser an entire tree. In bracket notation, the left tree is: $[S[Aa][B[Ba][Ba]]]$.

entire tree. In bracket notation, the left tree is: $[S[Aa][B[Ba]]]$.

The transform should thus be such, that the transformed grammar G' generates this entire string, including the brackets and the nonterminals. Moreover, G' should use the same number of productions to generate a tree t as the original grammar G needed to generate the parse, and with the same probabilities. Hence, for $S \rightarrow A\ B$ where G replaces the S, G' should generate a string that includes the brackets and the S:

$$S' \rightarrow [SA'B']$$

and similarly for all rules in the original grammar. The prime in S' (and A', B') is here used to distinguish between S as a terminal of G' (occurring in the string presented to the parser) and S' as a nonterminal of G'.

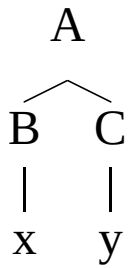
In BitPar, one additional difficulty is that it uses separate lexicon and grammar files, with a different format, and doesn't allow mixing terminal and nonterminals on the right-hand side of rules. The easiest way to deal with this is to create a unique, dummy nonterminal for every terminal symbol. For the grammar above, we then arrive at the following transformed grammar and dummy lexicon file:

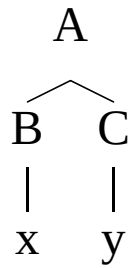
S" → [' S' A" B"]'	1.0
A" → [' A' A" A"]'	0.3
A" → [' A' a']'	0.7
B" → [' B' B" B"]'	0.8
B" → [' B' a']'	0.2

['	[
]'	]
S'	S
A'	A
B'	B
a'	a

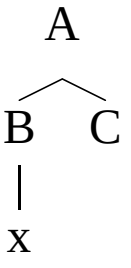
We can automatically generate the transformed grammar and dummy lexicon file from the original BitPar-grammar and lexicon using the unix utitilities 'sed' and 'awk'. Instead of the prime, we will use the symbol '@'.

First, transform the lexicon file format into the grammar file format (assuming, for the moment, a single preterminal per word, and replacing spaces within terminals):

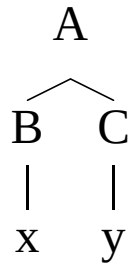
$A \circ$ 

$= A \circ [A[Bx][Cy]]$ 

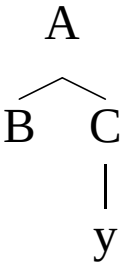
$A' \circ A' \to [A[Bx][Cy]] = ([A[Bx][Cy]])$

$A \circ$ 

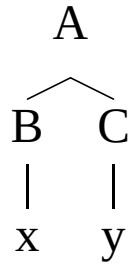
$\circ$ 

$= A \circ [A[Bx]C] \circ [Cy]$ 

$A' \circ A' \to [A[Bx]C'] \circ C' \to [Cy] = ([A[Bx]([Cy])])$

$A \circ$ 

$\circ$ 

$= A \circ [A[Bx]C] \circ [Bx]$ 

$A' \circ A' \to [AB'[Cy]] \circ B' \to [Bx] = ([A([Bx])[Cy]])$

- Treeparsing & Berkeley-style state-splitting (reranking)
- Combining DOP & Berkeley-style state-splitting (reranking)
- DMV + fold-unfold + DOP (Cohn et al., 2010)
- (Compositional) Distributional Semantics for reranking constituency parses (Le et al. 2013)