Better PCFGs

Miguel Ballesteros
7-11

Again some materials by Joakim Nivre.

Motivation

- Simply reading off a PCFG from a treebank is not likely to give the best possible parsing accuracy.

Problems with PCFGs

- The probability model associated with PCFG makes very rigid independence assumptions.
  - The probability of any rule expansion A → α is conditioned only on the LHS symbol A. It does not check the rest of the tree. → **Lack of context.**
  - The probability of any word (terminal symbol) is conditioned only on its own POS tag (preterminal symbols). It does not check the rest of the words in the tree.→ **Lack of lexical info.**

Problems with PCFGs

- **Lack of context.**

Tree context	NP PP	DT NN	PRP
Anywhere	11%	9%	6%
NP under S	9%	9%	21%
NP under VP	23%	7%	4%

This table gives statistics on the frequency of three rules for expanding NPs in English.

- the probability of an NP consisting only of a pronoun is underestimated in subject position (NP under S)
- while the probability of an NP containing a post-modifying PP is underestimated in object position (NP under VP).

Problems with PCFGs

- **Lack of context.**

Tree context	NP PP	DT NN	PRP
Anywhere	11%	9%	6%
NP under S	9%	9%	21%
NP under VP	23%	7%	4%

Two trees derived by exactly the same rules but in different order are

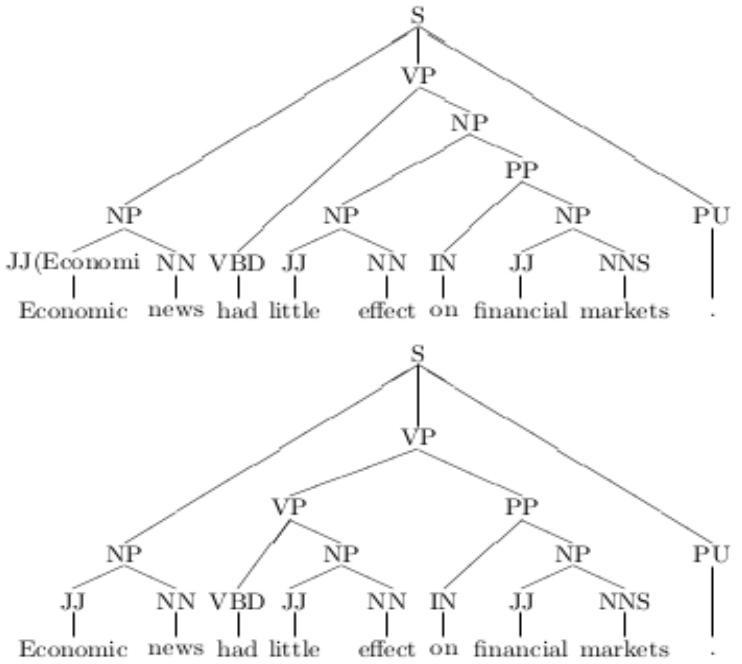
- *"The author of the book on the table"*
  - *"on the table"* can modify either "*the book"* or "*The author of the book"* **without any difference in probability.**

    Assuming that a rule NP → NP PP is used in both cases.

Problems with PCFGs

- **Lack of lexical info**

S	→	NP VP PU	1.00
VP	→	VP PP	0.33
VP	→	VBD NP	0.67
NP	→	NP PP	0.14
NP	→	JJ NN	0.57
NP	→	JJ NNS	0.29
PP	→	IN NP	1.00
PU	→	.	1.00
JJ	→	Economic	0.33
JJ	→	little	0.33
JJ	→	financial	0.33
NN	→	news	0.50
NN	→	effect	0.50
NNS	→	markets	1.00
VBD	→	had	1.00
IN	→	on	1.00



PP-Attachment problem.

The ranking of the two parse trees only depends on whether Q(NP → NP PP) > Q(VP → VP PP)

Insensitive to the lexical items involved

Problems with PCFGs

- **Lack of lexical info**

  - Statistics from the Penn Treebank show that a preposition like “into” Is almost 10 times more likely to attach to a verb than to a noun

  - While the preposition “of” is 100 times more likely to attach to a noun than a verb.

P

T

Q(NP → NP PP) > Q(VP → VP PP)

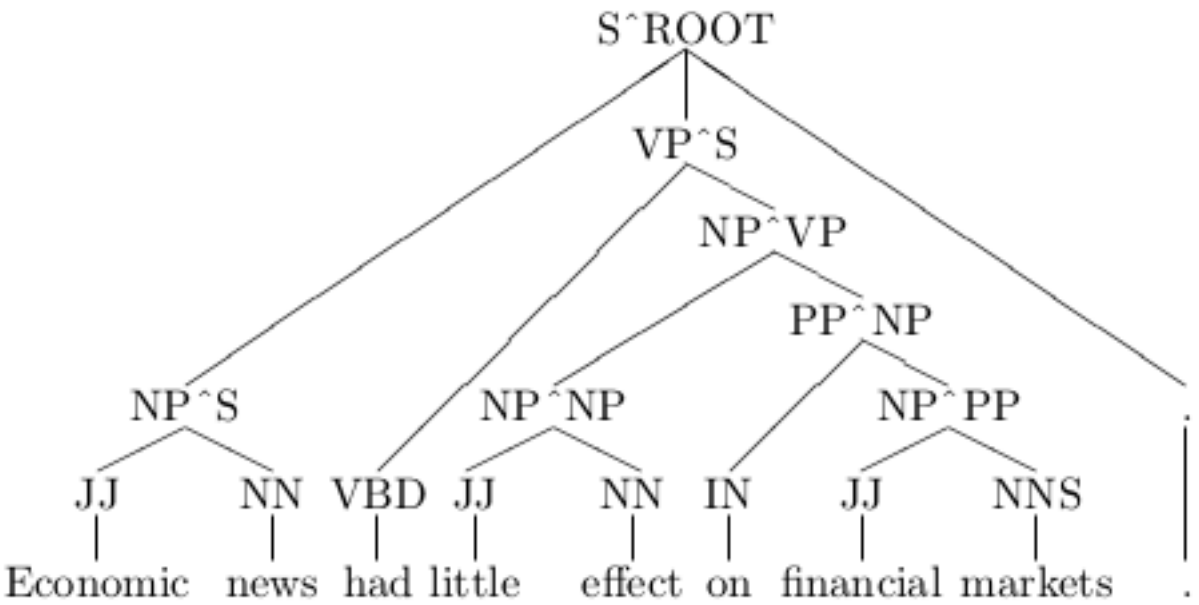
Insensitive to the lexical items involved

Problems with PCFGs

- Let's see some approaches to these problems ...

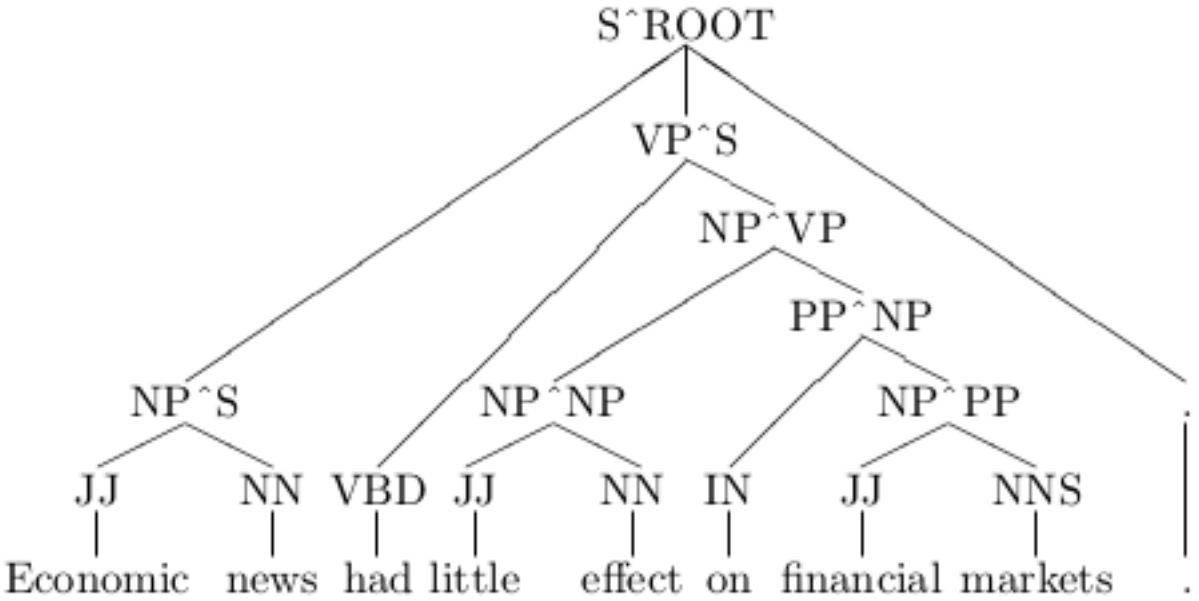
Parent Annotation

- Parent annotation (Johnson, 1998) is a simple technique for increasing the sensitivity to structural context.
- It consists in replacing each nonterminal A in a parse tree with AˆB, where B is the nonterminal on the parent node (in the untransformed tree).



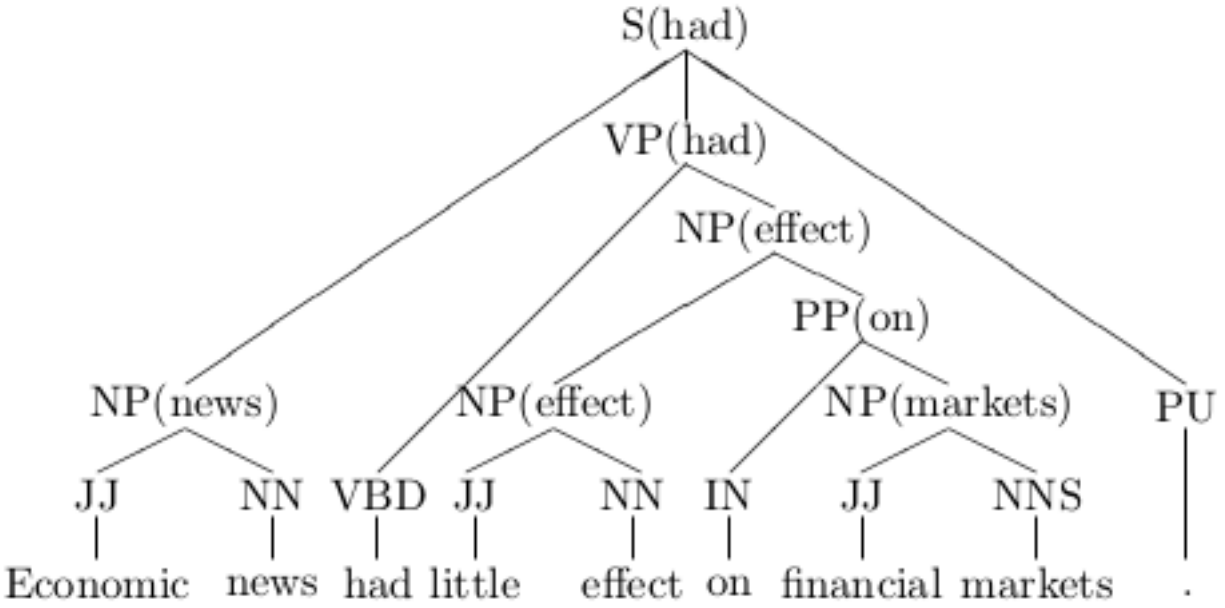
Parent Annotation

- Simple as it may seem, parent annotation often gives substantial improvements in accuracy compared to a plain treebank grammar.
- It can be generalized to grand-parent annotation (and so on), but this will increase the size of the grammar and may therefore lead to data sparsity in learning and inefficiency in decoding.



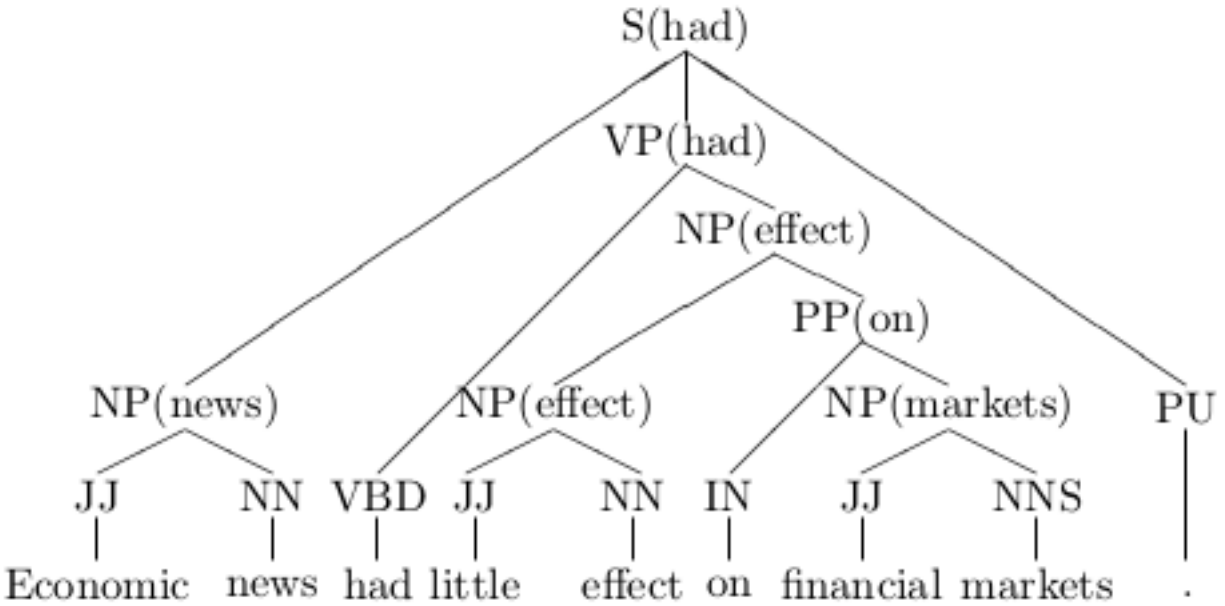
Lexicalization

- In a lexicalized PCFG, every nonterminal has the form A(a), where **A** is an ordinary nonterminal symbol and **a** is a terminal.
- In a head-lexicalized PCFG, we furthermore require that the right-hand side of every rule A(a) → α contains a nonterminal B(a) with the same lexical head as the parent category A(a).



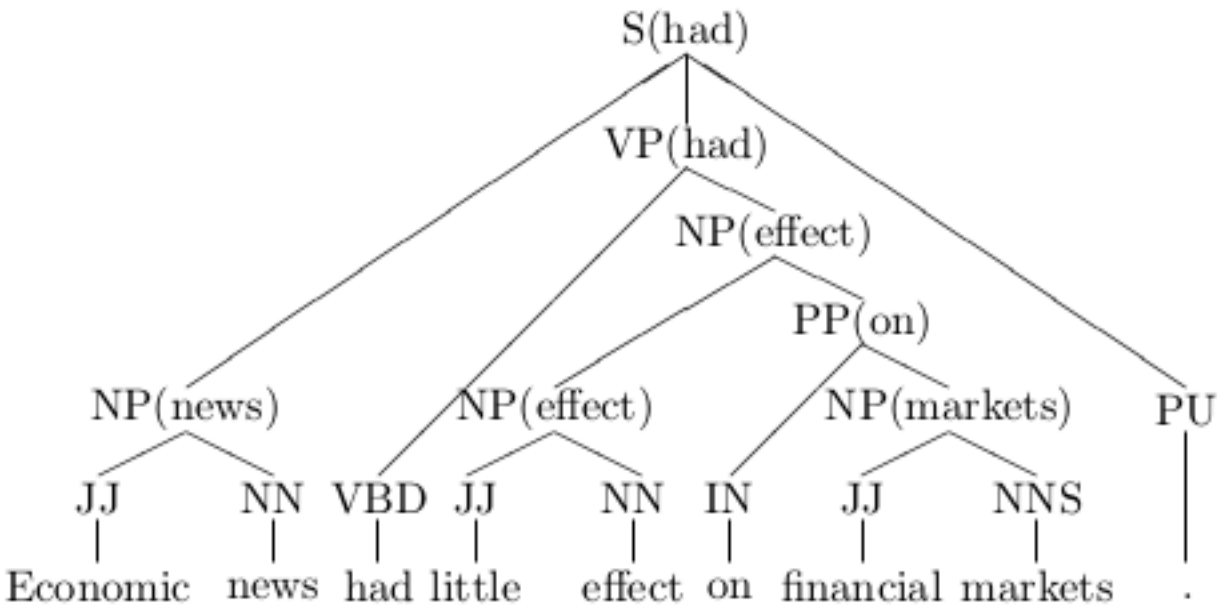
Lexicalization

- How can we find the head?



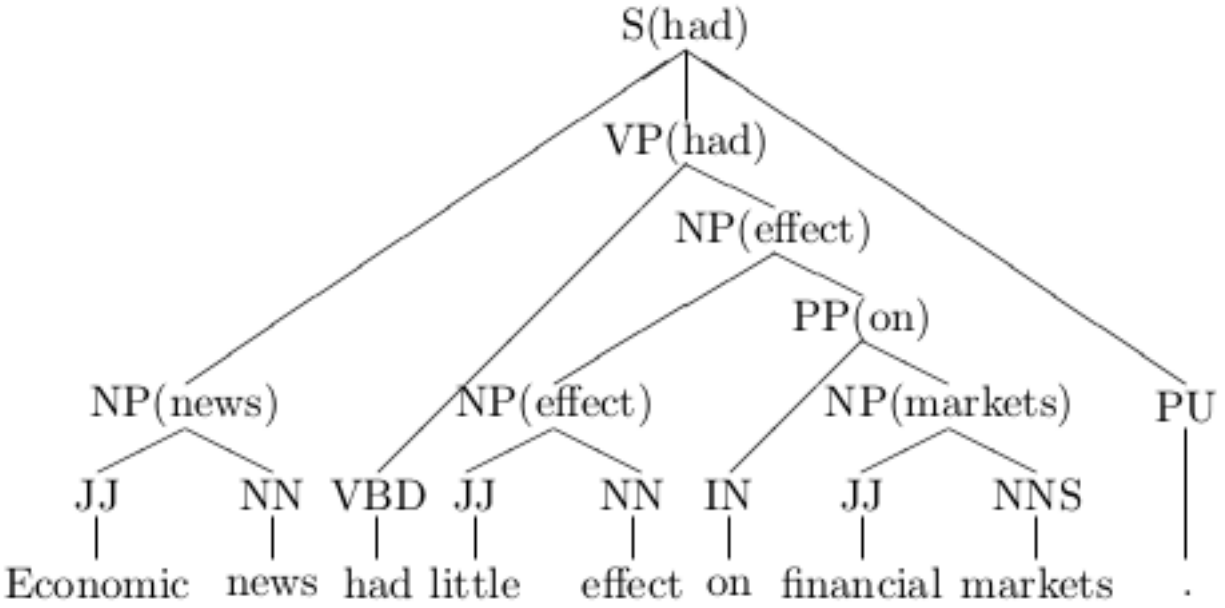
Lexicalization

- How can we find the head?
- Heuristic rules:
  - See head-finding rules by Collins (in his thesis).
  - Others.



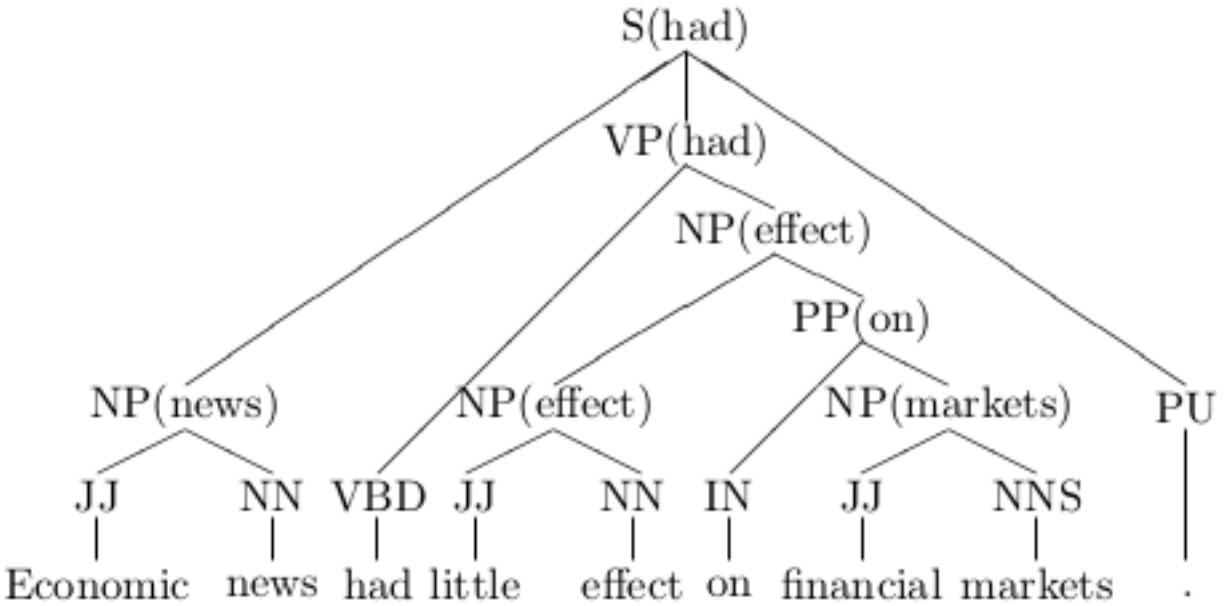
Lexicalization

- Lexicalization greatly increases the sensitivity to lexical information
  - Collins's parser (1997, 1999)
  - Charniak's parser (2000)
- F-score close to 90%.



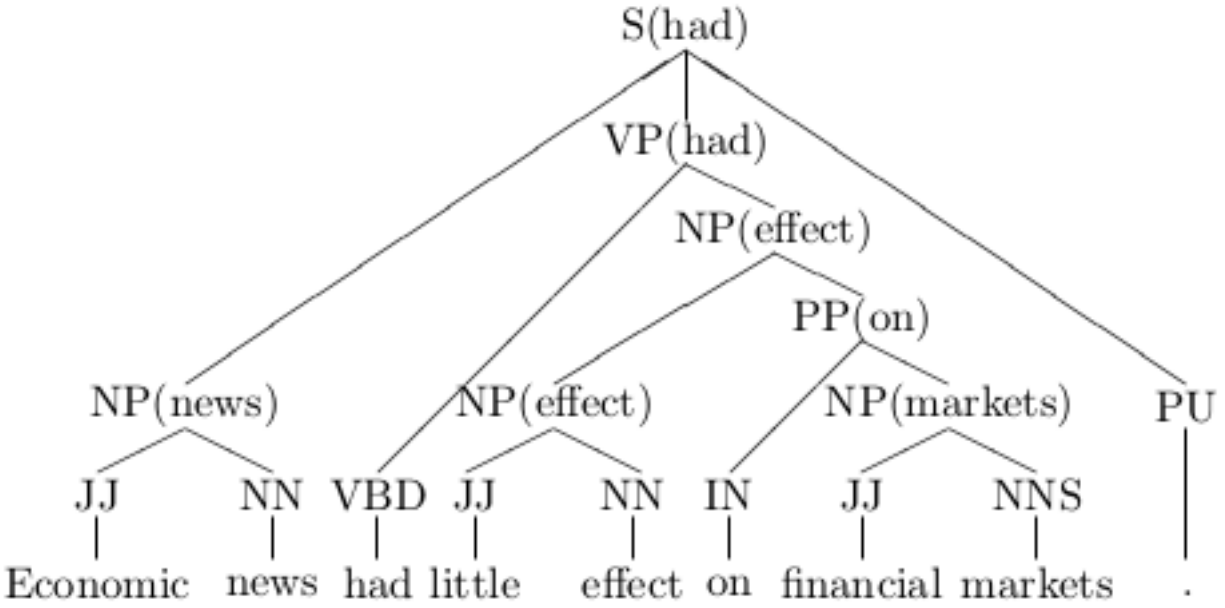
Lexicalization

- However, it also drastically increases the size of the grammar.
- Special care must be taken both in learning and in decoding.



Lexicalization

- **In learning**, maximum likelihood estimation based on raw relative frequencies performs poorly because of data sparsity
  - most lexicalized productions occur very rarely or not at all in the training corpus.
- It is necessary to apply some kind of smoothing, either
  - by backing off to unlexicalized rules or
  - by interpolating lexicalized and unlexicalized models



Lexicalization-Smoothing

- $Q = Q(A\ (a) \rightarrow B\ (b)\ C\ (a))$

  $= P\ (A \rightarrow_2 B\ C,\ b \mid A,a)$

  $= P\ (A \rightarrow_2 B\ C \mid A,a) \cdot P\ (b \mid A \rightarrow_2 B\ C,\ a)$

$P\ (A \rightarrow_2 B\ C \mid A,a) \approx$

$\lambda\ count\ (A \rightarrow_2 B\ C,\ a) / count\ (A,a) + (1 - \lambda)\ count\ (A \rightarrow_2 BC) / count\ (A)$

$P\ (b \mid A \rightarrow_2 B\ C,\ a) \approx$

$\lambda\ count\ (b, A \rightarrow_2 B\ C,\ a) / count\ (A \rightarrow_2 BC,\ a) + (1 - \lambda)\ count\ (b, A \rightarrow_2 BC) / count\ (A \rightarrow_2 BC)$

Lexicalization-Smoothing

- $Q = Q(A\ (a) \rightarrow B\ (b)\ C\ (a))$

  $= P\ (A \rightarrow_2 B\ C,\ b \mid A,a)$

  $= P\ (A \rightarrow_2 B\ C \mid A,a) \cdot P\ (b \mid A \rightarrow_2 B\ C,\ a)$

$P\ (A \rightarrow_2 B\ C \mid A,a) \approx$

$\lambda\ count\ (A \rightarrow_2 B\ C,\ a) / count\ (A,a) + (1 - \lambda)\ count\ (A \rightarrow_2 BC) / count\ (A)$

$P\ (b \mid A \rightarrow_2 B\ C,\ a) \approx$

$\lambda\ count\ (b, A \rightarrow_2 B\ C,\ a) / count\ (A \rightarrow_2 BC,\ a) + (1 - \lambda)\ count\ (b, A \rightarrow_2 BC) / count\ (A \rightarrow_2 BC)$

Probability of that the phrase A is rewritten as B and C, given that it has a lexical head a.

Lexicalization-Smoothing

- $Q = Q(A\ (a) \rightarrow B\ (b)\ C\ (a))$

  $= P\ (A \rightarrow_2 B\ C,\ b \mid A,a)$

  $= P\ (A \rightarrow_2 B\ C \mid A,a) \cdot P\ (b \mid A \rightarrow_2 B\ C,\ a)$

$P\ (A \rightarrow_2 B\ C \mid A,a) \approx$

$\lambda\ count\ (A \rightarrow_2 B\ C,\ a) / count\ (A,a) + (1 - \lambda)\ count\ (A \rightarrow_2 BC) / count\ (A)$

$P\ (b \mid A \rightarrow_2 B\ C,\ a) \approx$

$\lambda\ count\ (b, A \rightarrow_2 B\ C,\ a) / count\ (A \rightarrow_2 BC,\ a) + (1 - \lambda)\ count\ (b, A \rightarrow_2 BC) / count\ (A \rightarrow_2 BC)$

This is sometimes zero... this is why we use (1-λ)count(...) as in simple PCFGs.

Lexicalization-Smoothing

- $Q = Q(A\ (a) \rightarrow B\ (b)\ C\ (a))$

  $= P\ (A \rightarrow_2 B\ C,\ b \mid A,a)$

  $= P\ (A \rightarrow_2 B\ C \mid A,a) \cdot P\ (b \mid A \rightarrow_2 B\ C,\ a)$

$P\ (A \rightarrow_2 B\ C \mid A,a) \approx$

$\lambda\ count\ (A \rightarrow_2 B\ C,\ a) / count\ (A,a) + (1 - \lambda)\ count\ (A \rightarrow_2 BC) / count\ (A)$

$P\ (b \mid A \rightarrow_2 B\ C,\ a) \approx$

$\lambda\ count\ (b, A \rightarrow_2 B\ C,\ a) / count\ (A \rightarrow_2 BC,\ a) + (1 - \lambda)\ count\ (b, A \rightarrow_2 BC) / count\ (A \rightarrow_2 BC)$

Probability that b is the lexical head of the non-head child (B).

Lexicalization-Smoothing

- $Q = Q(A\ (a) \rightarrow B\ (b)\ C\ (a))$

  $= P\ (A \rightarrow_2 B\ C,\ b \mid A,a)$

  $= P\ (A \rightarrow_2 B\ C \mid A,a) \cdot P\ (b \mid A \rightarrow_2 B\ C,\ a)$

$P\ (A \rightarrow_2 B\ C \mid A,a) \approx$

$\lambda\ count\ (A \rightarrow_2 B\ C,\ a) / count\ (A,a) + (1 - \lambda)\ count\ (A \rightarrow_2 BC) / count\ (A)$

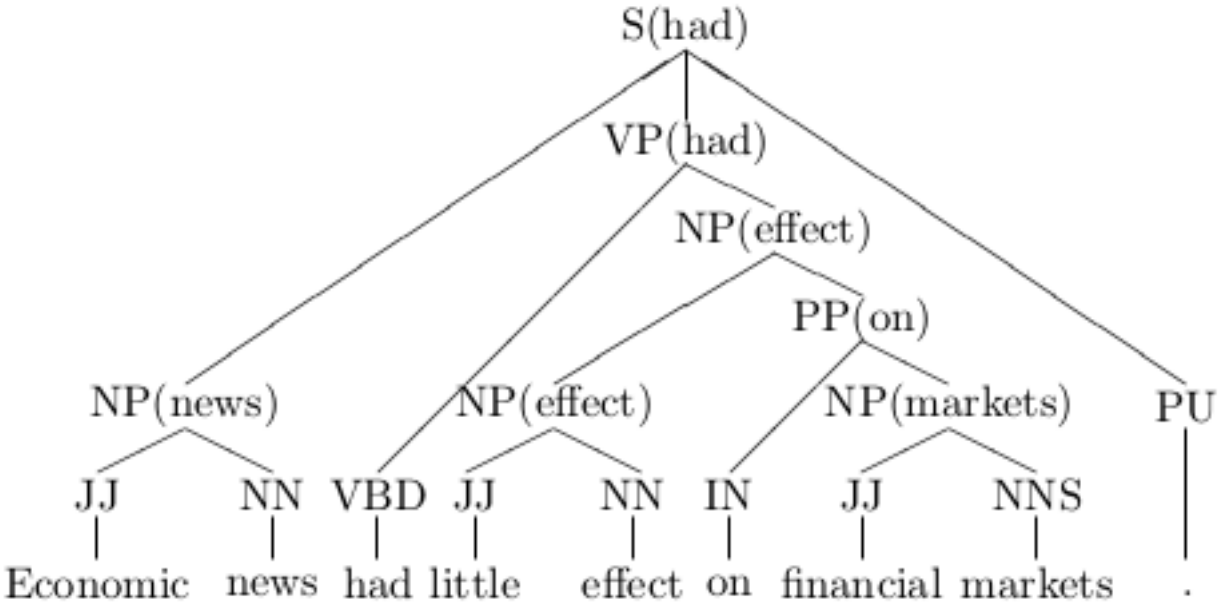
$P\ (b \mid A \rightarrow_2 B\ C,\ a) \approx$

$\lambda\ count\ (b, A \rightarrow_2 B\ C,\ a) / count\ (A \rightarrow_2 BC,\ a) + (1 - \lambda)\ count\ (b, A \rightarrow_2 BC) / count\ (A \rightarrow_2 BC)$

If this is zero, we need to also use (1-λ) (count …), in which we do Not care about the lexical head a.

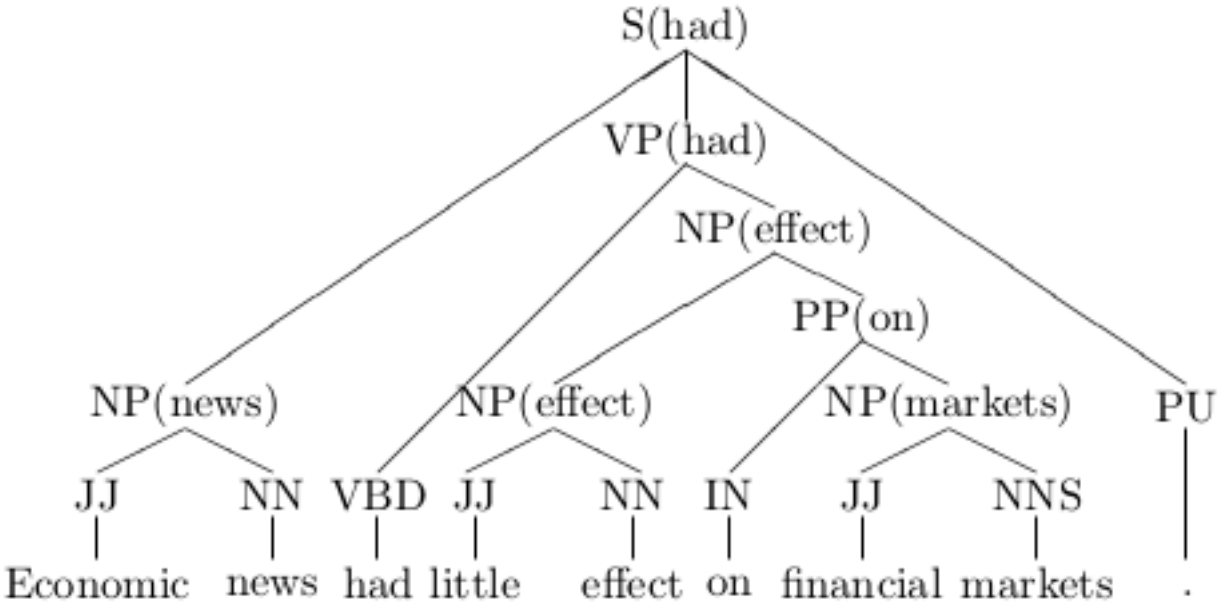
Lexicalization

- **In decoding**, the increased size of the grammar affects running time negatively.
- We need two additional loops, one over lexical heads in the head child and one over lexical heads in the non-head child (assuming a binarized grammar), both of which take time that grows linearly with sentence length.
- This means that the time complexity of straightforward CKY parsing is **$O(n^5)$** (instead of $O(n^3)$ in the unlexicalized case);



Lexicalization

- **In decoding**, ...
- It is therefore often necessary to prune the search space, which means giving up the theoretical guarantee of finding the Viterbi parse.



Lexicalization

- This technique, simple as it may seem, improves the accuracy a lot.
- From 75.x F1 we get by using the technique known as “reading-off” a grammar (see previous lecture)
- We reach 90.x F1.
- See the influential parsers by Michael Collins and Eugene Charniak.

Markovization

- Markovization is a technique for splitting n-ary grammar rules into sets of simpler rules that generate one child at a time.
  - conditioned on a limited number of siblings
  - making the grammar more robust to missing rules
  - **counteracting data sparsity in case of a lexicalized grammar.**

Markovization

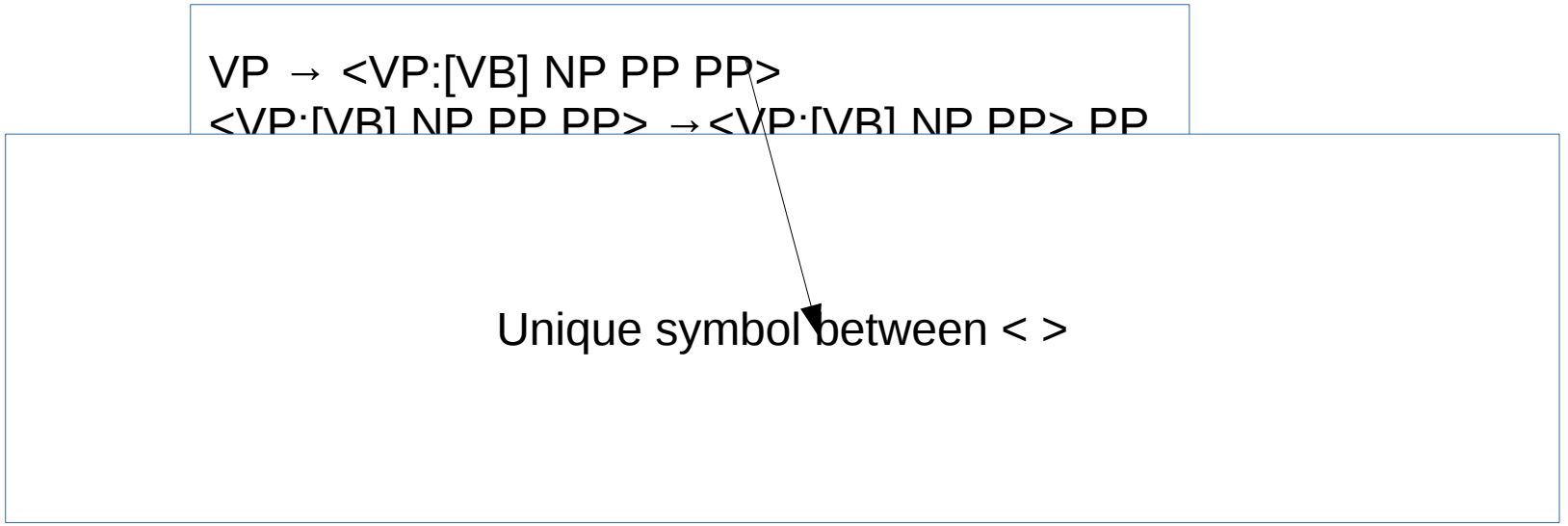
- For example, the rule
  - VP → VB NP PP PP can be binarized exactly as:

VP → <VP:[VB] NP PP PP>
<VP:[VB] NP PP PP> →<VP:[VB] NP PP> PP
<VP:[VB] NP PP> → <VP:[VB] NP> PP
<VP:[VB] NP> → <VP:[VB]> NP
<VP:[VB]> → VB

This is (like) transforming to CNF!

Markovization

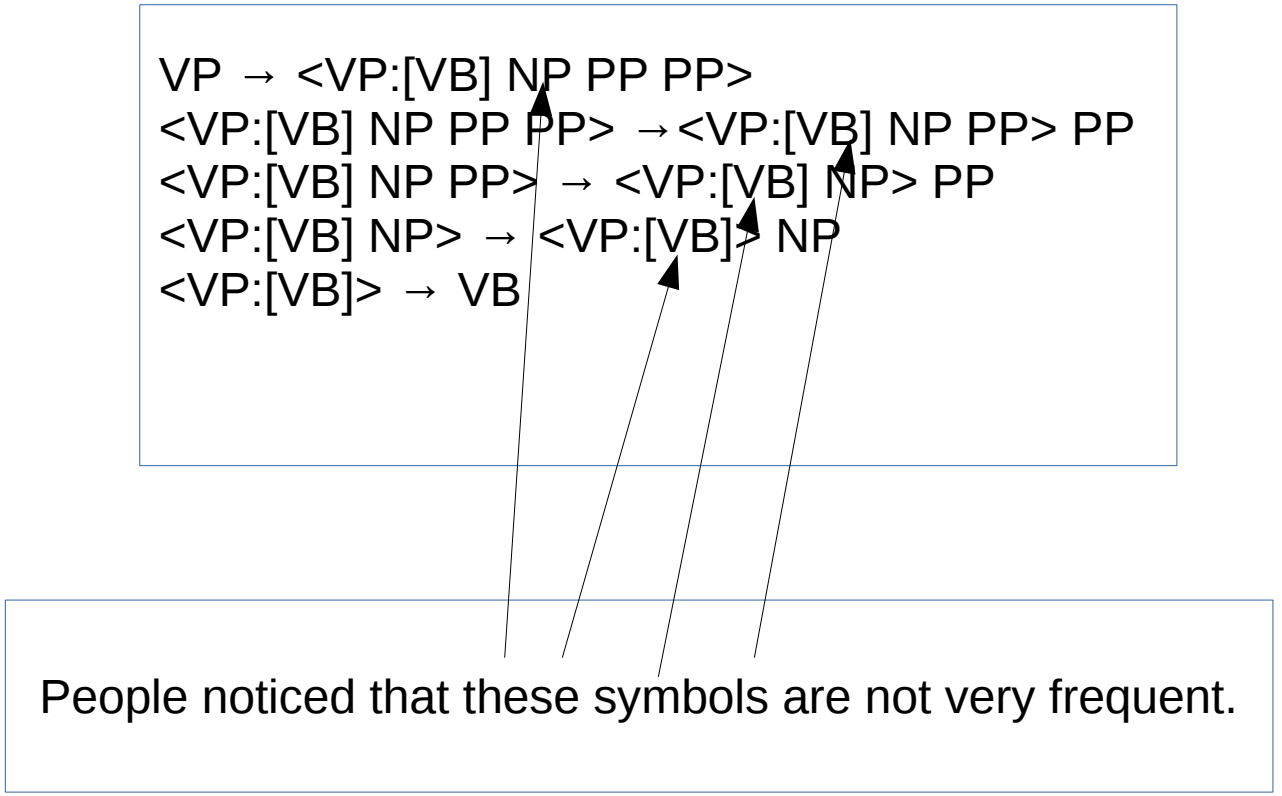
- For example, the rule
  - VP → VB NP PP PP can be binarized exactly as:



This is (like) transforming to CNF!

Markovization

- For example, the rule
  - VP → VB NP PP PP can be binarized exactly as:



Markovization

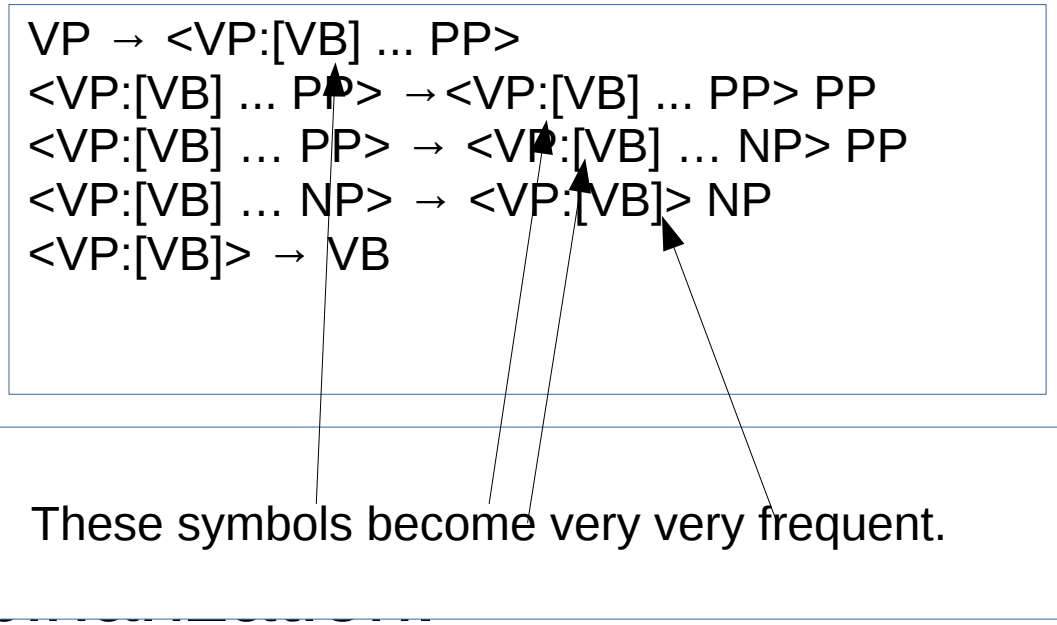
- In first-order markovization, we only remember the immediately preceding sibling for every child
  - VP → VB NP PP PP as:

VP → <VP:[VB] ... PP>
<VP:[VB] ... PP> →<VP:[VB] ... PP> PP
<VP:[VB] … PP> → <VP:[VB] … NP> PP
<VP:[VB] … NP> → <VP:[VB]> NP
<VP:[VB]> → VB

- Markovization can be therefore understood as forgetful binarization.

Markovization

- In first-order markovization, we only remember the immediately preceding sibling for every child
  - VP → VB NP PP PP as:



- Marko… …ood as forget…

Markovization

- This is sometimes referred to as horizontal markovization, because it determines the amount of horizontal context taken into account when generating a child.
- **Generalized parent annotation can be understood as vertical markovization.**

  – VP → VB NP PP PP as:

VP → <VP:[VB] ... PP>
<VP:[VB] ... PP> →<VP:[VB] ... PP> PP
<VP:[VB] … PP> → <VP:[VB] … NP> PP
<VP:[VB] … NP> → <VP:[VB]> NP
<VP:[VB]> → VB

Markovization

- A plain treebank PCFG (previous lecture) has first-order vertical markovization and infinite-order horizontal markovization, neither of which is really optimal.
  - The former is too restrictive and makes the grammar insensitive to structural context.
  - The latter is too expressive and may cause data sparsity and lack of robustness.

    (Klein and Manning, 2003)

Latent Variables

- As we have seen, many of the recent improvements in PCFG parsing come from techniques for transforming a plain treebank grammar to a grammar that is better suited for parsing.
  - This involves replacing the coarse-grained categories in the treebank by more fine-grained categories,
  - This is known as **state splitting**.
- Both parent annotation and lexicalization can be seen as instances of **state splitting**:
  - Parent annotation splits a nonterminal category A into one subcategory for every nonterminal that can be found as a parent of A,
  - Lexicalization splits A into one subcategory for every terminal that can be the lexical head of a phrase of type A.

Latent Variables

- Drawback:These two approaches split all categories to the same level of granularity.
  - In the case of lexicalization can sometimes be harmful because we simply end up with too many categories and parameters to estimate.
  - We have markovization... yes, but it can be better.

Latent Variables

- A small set of manually defined state splits can help to increase the sensitivity of the grammar without leading to data sparsity. (Klein & Manning, 2003)
  - But... this takes “manual” time.

- Petrov et al. (2006) showed how to use **latent variables** to learn good state splits with an unlexicalized parser to reach the same level of accuracy (as the best lexicalized PCFG parsers).

Latent Variables

- Latent variable: to replace each nonterminal A observed in the treebank by a set of categories $A[X_1], \ldots, A[X_n]$ defined by latent (or ***unobserved***) variables $X_1 \ldots X_n$.
- We then duplicate all rules in the treebank grammar.
  - so that we have distinct copies for every new instantiation of the old category.
  - We use EM to learn probabilities for the rules of our new expanded grammar.
    - EM is very constrained in this case (than in “learning without a treebank”) because a rule with the latent variable $A[X_i]$ can only be used in case the original treebank tree contains the symbol A.

Latent Variables

- The key to getting high parsing accuracy with latent variable grammars is to find the right level of granularity
  - (that is, the right number of variables) for a given category.

  Petrov et al. (2006) use an iterative split-merge strategy, which performs
  - successive binary splits only as long as parsing accuracy improves.
  - merges splits that do not improve parsing accuracy.