TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2019

Lecture 9:
Inference in Structured Prediction

Roadmap

- intro (1 lecture)
- deep learning for NLP (5 lectures)
- **structured prediction (4 lectures)**
  - introducing/formalizing structured prediction, categories of structures
  - **inference: dynamic programming, greedy algorithms, beam search**
  - **inference with non-local features**
  - learning in structured prediction
- generative models, latent variables, unsupervised learning, variational autoencoders (2 lectures)
- Bayesian methods in NLP (2 lectures)
- Bayesian nonparametrics in NLP (2 lectures)
- review & other topics (1 lecture)

Assignments

- Assignment 2 due Wednesday
- for the report, please use either pdf format or a Jupyter notebook (no plain text)

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\operatorname{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y}} \ \operatorname{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Working definition of structured prediction:

size of output space is exponential in size of input

or is unbounded (e.g., machine translation)

(we can't just enumerate all possible outputs)

Inference with Structured Predictors

inference: solve argmax

$$\text{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \ \text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

- how do we efficiently search over the space of all structured outputs?
- this space may have size exponential in the size of the input, or be unbounded
- complexity of inference depends on parts function

Parts and Score Functions

– given a "parts" function

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y})$$

– our score function is then defined:

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \text{parts}(\boldsymbol{x}, \boldsymbol{y})} \text{score}_{\text{part}}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

– each part is a subcomponent of input/output pair

– score function decomposes additively across parts

Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
unlabeled dependency parsing	tree over the words in the input sentence; each word has exactly one parent	set containing indices of parent words for each word in sentence	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{0, 1, \ldots, T\}$
conditional generation	sentence (or a paragraph, document, etc.)	set containing each word in the output	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_{\lvert\boldsymbol{y}\rvert}\}$ where each $y_t \in \mathcal{V}$

Hidden Markov Model (HMM)

$$p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = p_{\boldsymbol{\tau}}(\text{</s>} \mid y_{|\boldsymbol{x}|}) \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

transition parameters: $p_{\boldsymbol{\tau}}(y_i \mid y_{i-1})$

emission parameters: $p_{\boldsymbol{\eta}}(x_i \mid y_i)$

$$\text{parts}_{\text{HMM}}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle x_t, y_t \rangle\}_{t=1}^{T} \cup \{\langle \emptyset, y_{t-1:t} \rangle\}_{t=1}^{T}$$

- each word-label pair forms a part, and each label bigram forms a part
- note: define score as log-probability to make score function decompose additively over parts

Inference in HMMs

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{\boldsymbol{y}}{\text{argmax}}\ p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$$

$$= \underset{\boldsymbol{y}}{\text{argmax}}\ p_{\boldsymbol{\tau}}(</\text{s}> \mid y_{|\boldsymbol{x}|}) \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

- since the output is a sequence, this argmax requires iterating over an exponentially-large set
- we can use **dynamic programming (DP)** to solve these problems exactly
- for HMMs (and other sequence models), the algorithm for solving this is the **Viterbi algorithm**

Viterbi Algorithm for HMMs

- recursive equations + memoization:

base case:
returns probability of sequence starting with label *y* for first word

$$V(1, y) = p_{\boldsymbol{\eta}}(x_1 \mid y)\, p_{\boldsymbol{\tau}}(y \mid \text{<s>})$$

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

recursive case:
computes probability of max-probability label sequence that ends with label *y* at position *m*

final value is in: $goal(\boldsymbol{x}) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\tau}}(\text{</s>} \mid y')\, V(|\boldsymbol{x}|, y') \right)$

"Backpointers" in Viterbi

- Viterbi only gives us the probability of the max-probability label sequence
- how do we get the actual label sequence?

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

$$L(m, y) = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

contains label that achieved max probability in max-prob label sequence that ends with label *y* at position *m*

“Backpointers” in Viterbi

- Viterbi only gives us the probability of the max-probability label sequence
- how do we get the actual label sequence?

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

$$L(m, y) = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

similar modification for final label:

$$goal(\boldsymbol{x}) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\tau}}(\text{</s>} \mid y')\, V(|\boldsymbol{x}|, y') \right)$$

$$\hat{y}_{|\boldsymbol{x}|} = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\tau}}(\text{</s>} \mid y')\, V(|\boldsymbol{x}|, y') \right)$$

Following Backpointers in Viterbi

- full backpointer-following procedure (after Viterbi):

$$1 < m \leq |\boldsymbol{x}| : \quad L(m, y) = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y) \, p_{\boldsymbol{\tau}}(y \mid y') \, V(m-1, y') \right)$$

$$\hat{y}_{|\boldsymbol{x}|} = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\tau}}(</\mathrm{s}> \mid y') \, V(|\boldsymbol{x}|, y') \right)$$

$$\begin{aligned} \text{let } T = |\boldsymbol{x}| : \quad & \hat{y}_T \\ & \hat{y}_{T-1} = L(T, \hat{y}_T) \\ & \hat{y}_{T-2} = L(T-1, \hat{y}_{T-1}) \\ & \ldots \\ & \hat{y}_2 = L(3, \hat{y}_3) \\ & \hat{y}_1 = L(2, \hat{y}_2) \end{aligned}$$

Viterbi Algorithm for Sequence Models

(with tag bigram features)

$$V(1, y) = \text{score}(\boldsymbol{x}, \langle \text{<s>}, y \rangle, 1, \boldsymbol{w})$$

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, \langle y', y \rangle, m, \boldsymbol{w}) + V(m-1, y') \right)$$

score function for label bigram <*y’*, *y*> ending at position *m* in ***x***

could be anything! linear model, feed-forward network, LSTM, etc.

Approximate Inference

- exact inference limits us to small parts functions
  - e.g., Viterbi requires parts with only two consecutive labels, and takes time $O(|x|\ |L|^2)$
  - time complexity of exact DP algorithms is exponential in the size of the parts
- we want to use bigger parts without exponential increase in runtime
- so, we consider algorithms for **approximate inference**
- even when using small parts, approximate inference can help us to speed up inference with little loss in accuracy

Example: HMM POS Tagging

- tag set:

  N: noun

  V: verb

  D: determiner

  J: adjective

  example sentence:

V	D	N
Lower	the	lights

Greedy Left-to-Right Inference

- build a label sequence one word at a time from left to right
- at each position, choose the tag for a word greedily to maximize a local scoring function

Greedy Inference

Lower the lights

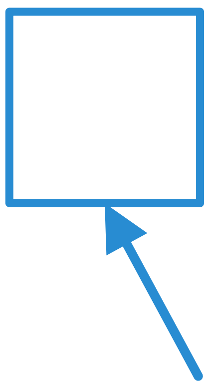
<s>

starting tag

Greedy Inference

Lower the lights

<s>



$$\hat{y}_1 = \operatorname*{argmax}_{y \in \mathcal{L}} \; p_{\boldsymbol{\eta}}(\text{“Lower”} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \text{<s>})$$

takes O(|*L*|) time (iterate through labels)

Greedy Inference

Lower the lights

<s> J

$$\hat{y}_1 = \operatorname*{argmax}_{y \in \mathcal{L}} \; p_{\boldsymbol{\eta}}(\text{"Lower"} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \texttt{<s>})$$

error here: model must choose a tag and stick with it; can’t change anything later

V D N

Lower the lights

Greedy Inference

Lower the lights

<s> J D

$$\hat{y}_2 = \operatorname*{argmax}_{y \in \mathcal{L}} \; p_{\boldsymbol{\eta}}(\text{“the”} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \text{J})$$

uses best label for previous position

V D N

Lower the lights

Greedy Inference

Lower the lights

<s> J D N </s>

$$\hat{y}_3 = \operatorname*{argmax}_{y \in \mathcal{L}} \; p_{\boldsymbol{\eta}}(\text{“lights”} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \texttt{D})\, p_{\boldsymbol{\tau}}(\texttt{</s>} \mid y)$$

for final word in input, include transition to end-of-sentence label

V D N

Lower the lights

Greedy Inference

Lower the lights

<s> J D N </s>

$|\boldsymbol{x}|$ positions, $O(|L|)$ time for each position (iterate through labels)

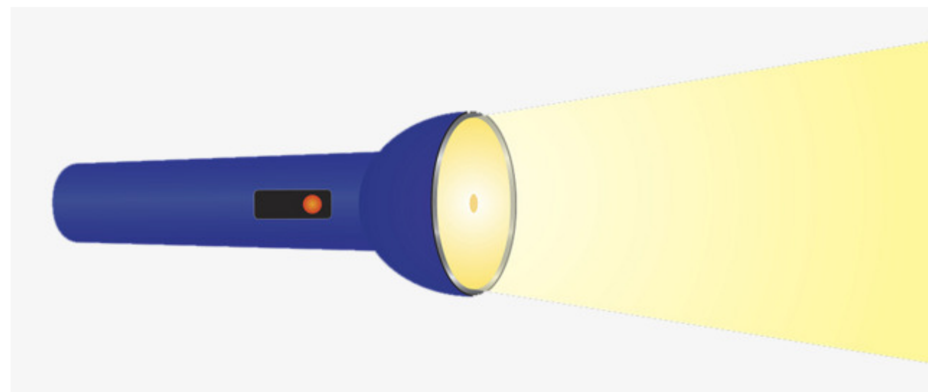
time for greedy: $O(|\boldsymbol{x}|\ |L|)$

time for Viterbi: $O(|\boldsymbol{x}|\ |L|^2)$

faster than Viterbi, but doesn’t work as well
can we improve it?

- we can convert this greedy algorithm to **beam search**

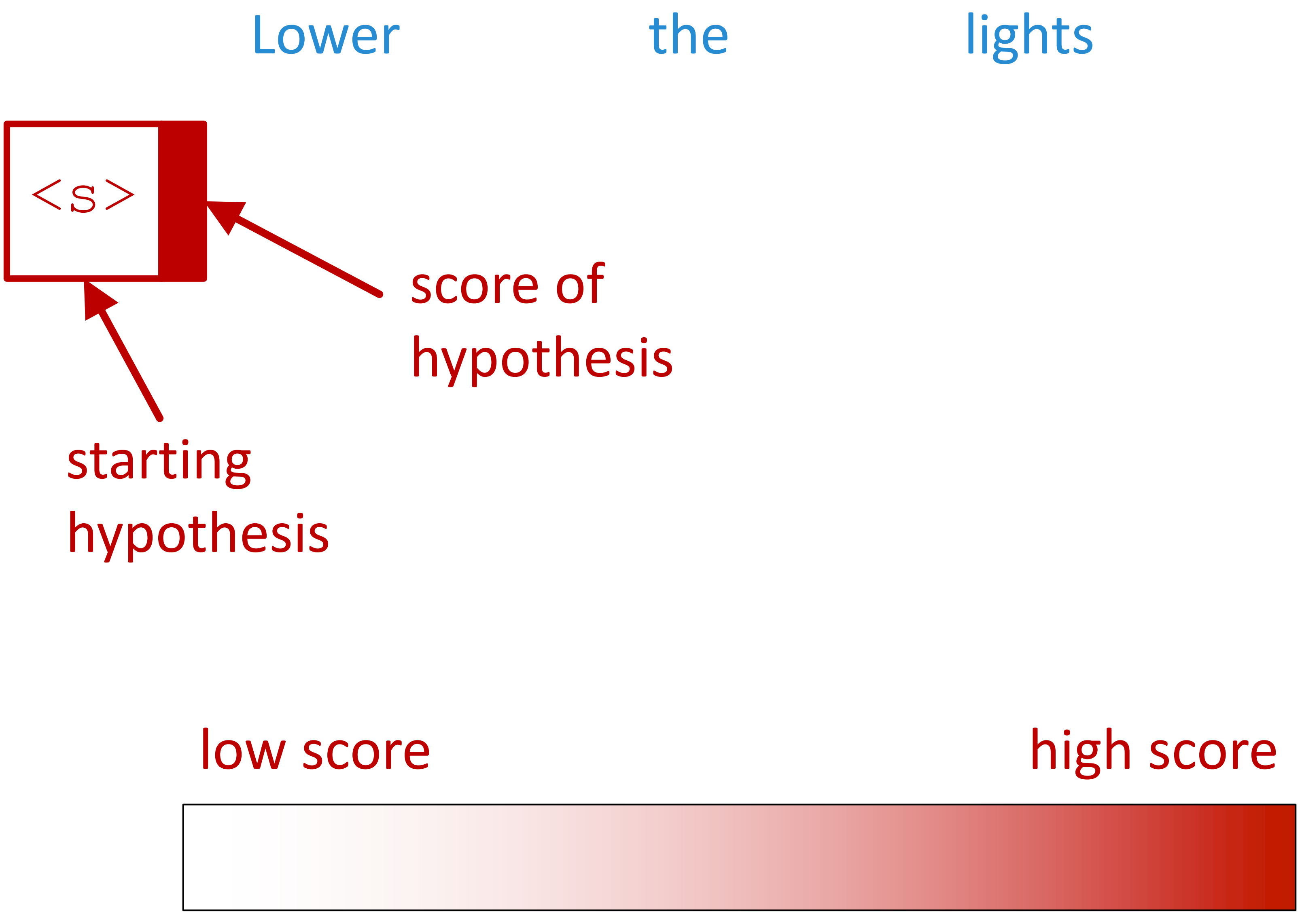
- beam search maintains multiple hypotheses at each position



- two types of steps:
  - extend hypotheses
  - prune set of hypotheses

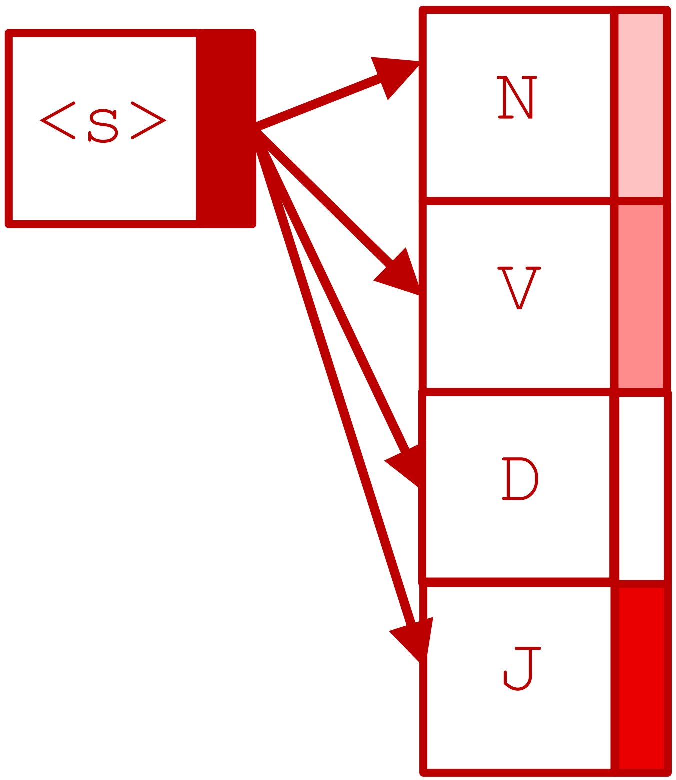
- size of pruned set = size of “beam”

Beam Search (beam size b = 2)



Extend Hypotheses

Lower the lights



only one hypothesis here to extend

consider all possible ways of extending it

scores of extended hypotheses:

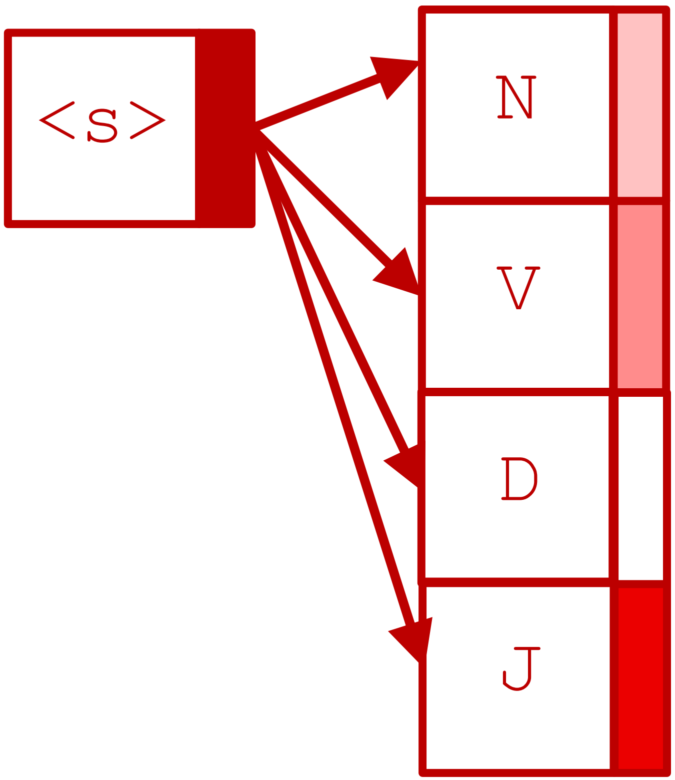
$$p_{\boldsymbol{\eta}}(\text{“Lower”} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \text{<s>})\, \underbrace{\text{score}(\text{hyp}_{\text{prev}})}$$

score of previous hypothesis

low score high score

Extend Hypotheses

Lower the lights



only one hypothesis here to extend

consider all possible ways of extending it

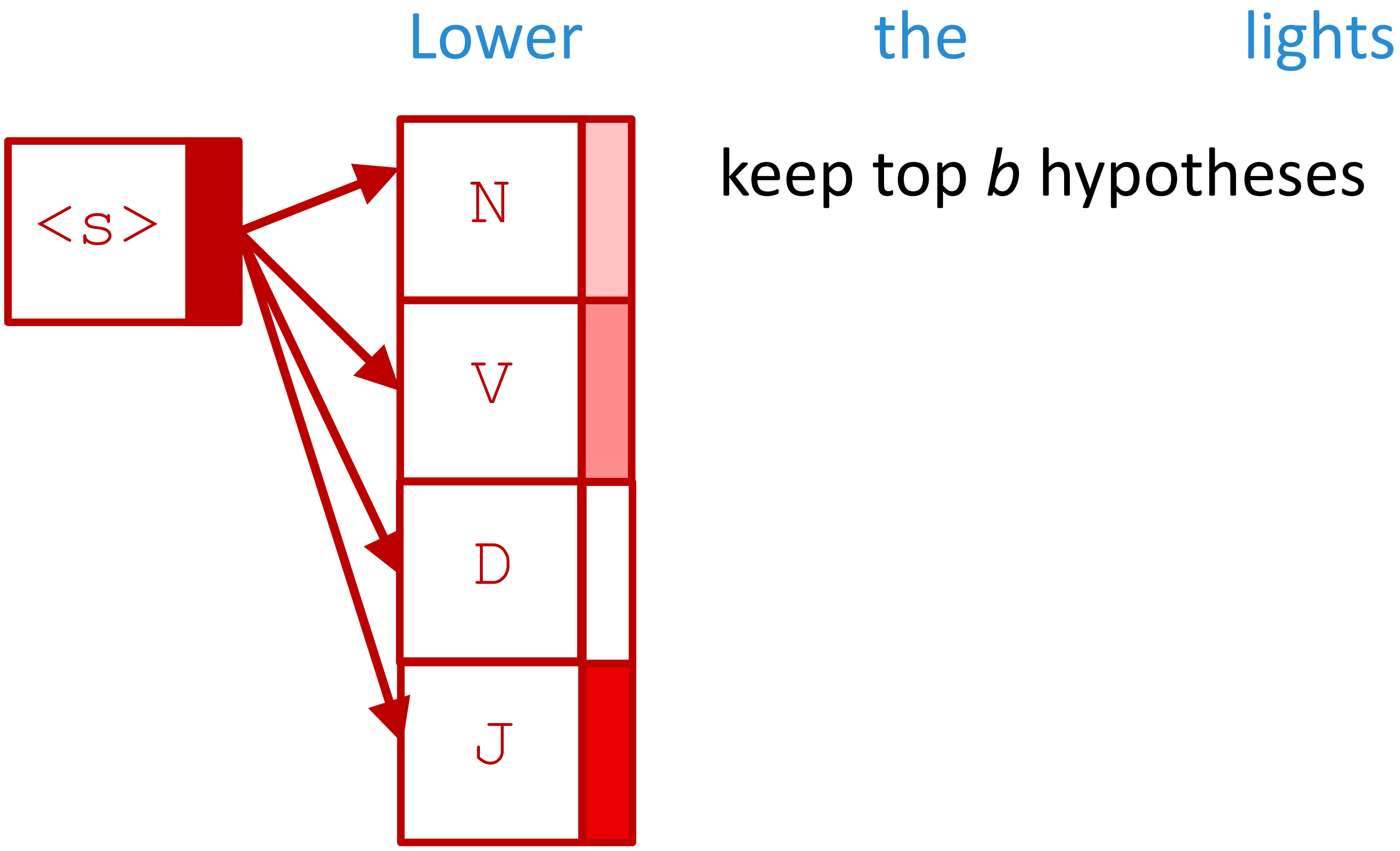
scores of extended hypotheses:

$$p_{\boldsymbol{\eta}}(\text{“Lower”} \mid y)\, p_{\boldsymbol{\tau}}(y \mid \texttt{<s>})$$

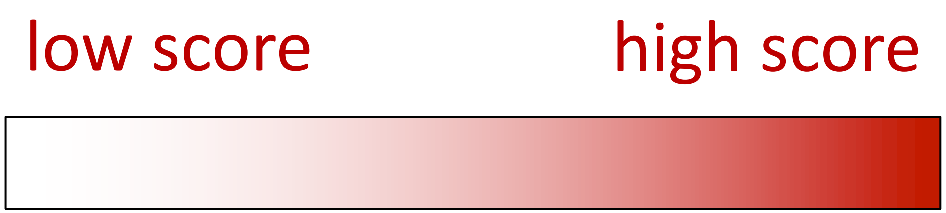
because score of starting hypothesis is fixed to 1

low score high score

Prune Hypotheses (b = 2)

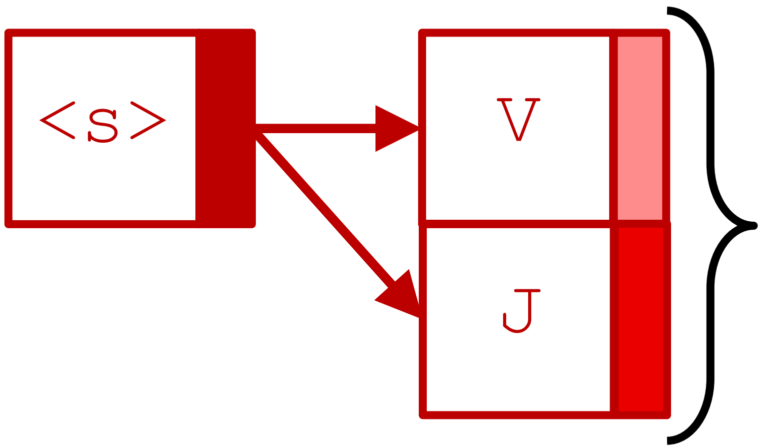


keep top *b* hypotheses



Prune Hypotheses (b = 2)

Lower the lights



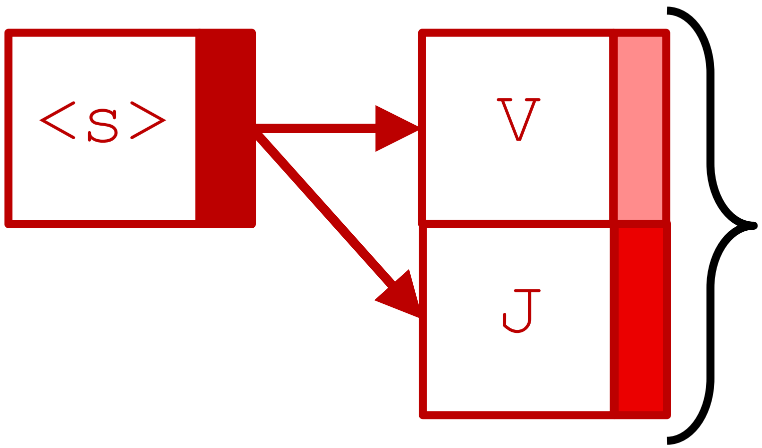
$B(1)$ = set containing the top *b* hypotheses ending at position 1, along with their scores

note: this is a set; the items do **not** have to be sorted

low score high score

Prune Hypotheses (b = 2)

Lower the lights



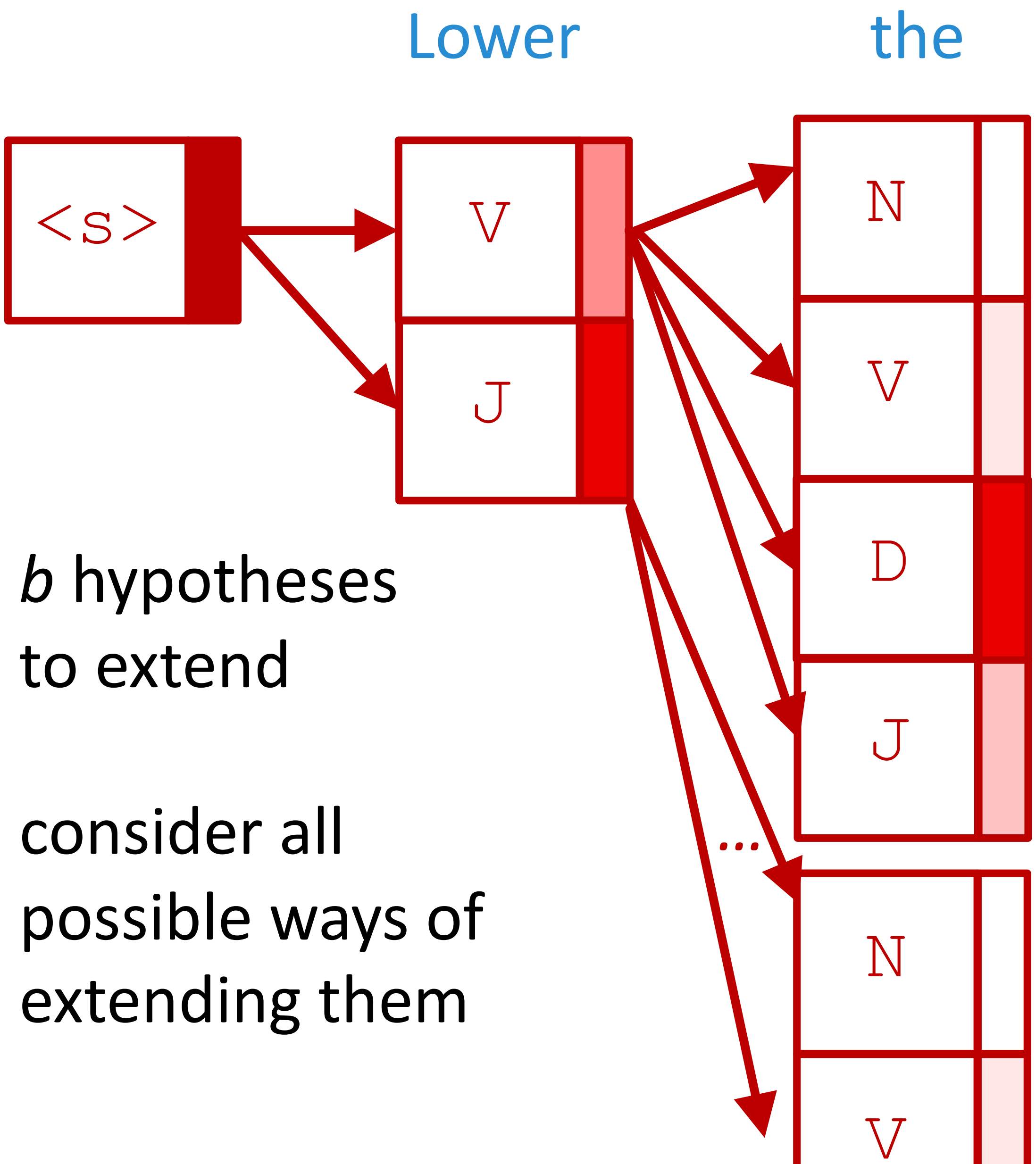
$B(1)$ = set containing the top *b* hypotheses ending at position 1, along with their scores

note: this is a set; the items do **not** have to be sorted

so, this step only takes O(*N*) time if there are *N* hypotheses to sort; cf. “unordered partial sorting”

low score high score

Extend Hypotheses



b hypotheses to extend

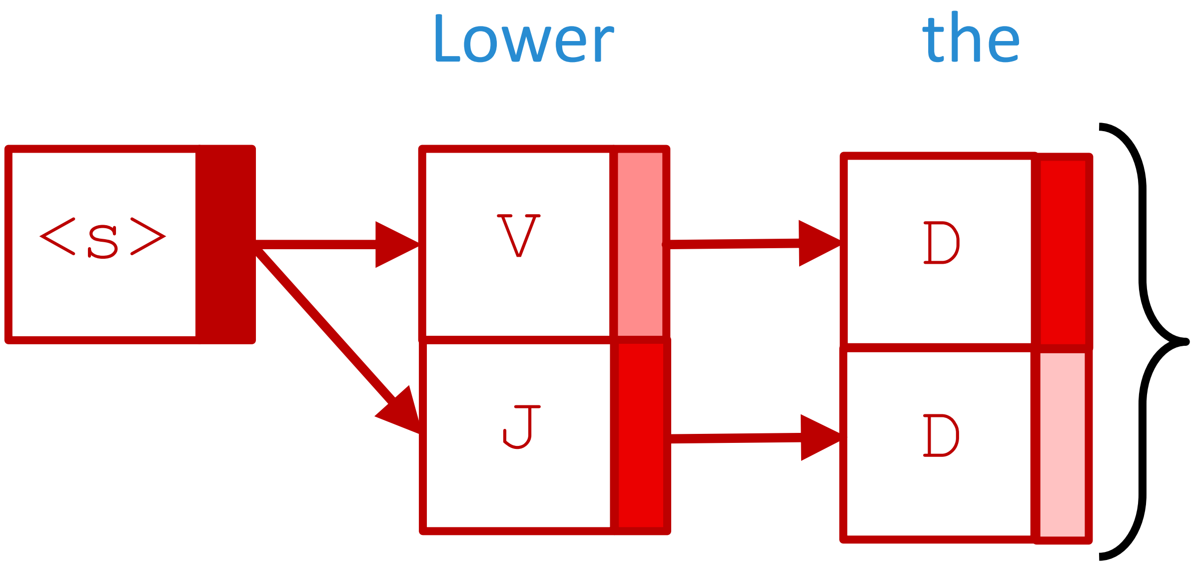
consider all possible ways of extending them

scores of extended hypotheses:

$$p_{\boldsymbol{\eta}}(\text{“the”} \mid y) \times p_{\boldsymbol{\tau}}(y \mid y_{\text{prev}}) \times \text{score}(\text{hyp}_{\text{prev}})$$

last label from previous hypothesis

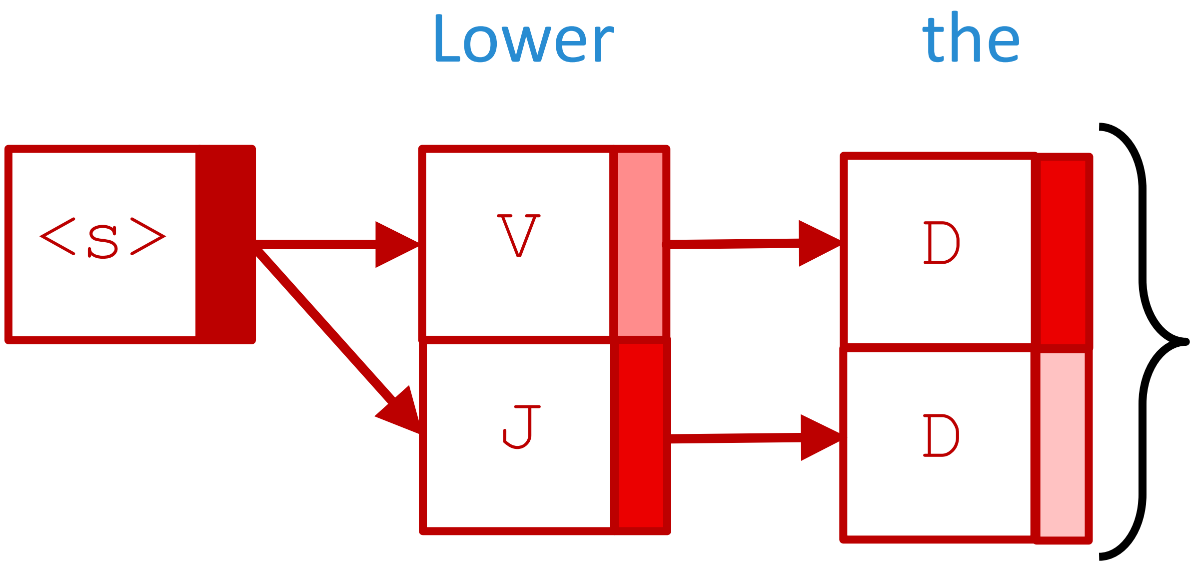
Prune Hypotheses (b = 2)



$B(2)$ = set containing the top *b* hypotheses ending at position 2, with their scores

note: using backpointers, we can recover the entire hypothesis

Prune Hypotheses (*b* = 2)



$B(2)$ = set containing the top *b* hypotheses ending at position 2, with their scores

computational complexity of beam search?

Complexity of Beam Search

$|\boldsymbol{x}|$ positions

extend hypotheses:
$O(b|L|)$ time for each position ($O(b)$ hypotheses, for each we have to iterate through labels)

prune set of hypotheses:
$O(b|L|)$ time for each position (unordered partial sorting takes $O(N)$ time for a set with N items)

time for beam: $O(|\boldsymbol{x}|\ b|L|)$
time for greedy: $O(|\boldsymbol{x}|\ |L|)$
time for Viterbi: $O(|\boldsymbol{x}|\ |L|^2)$

Beam Search

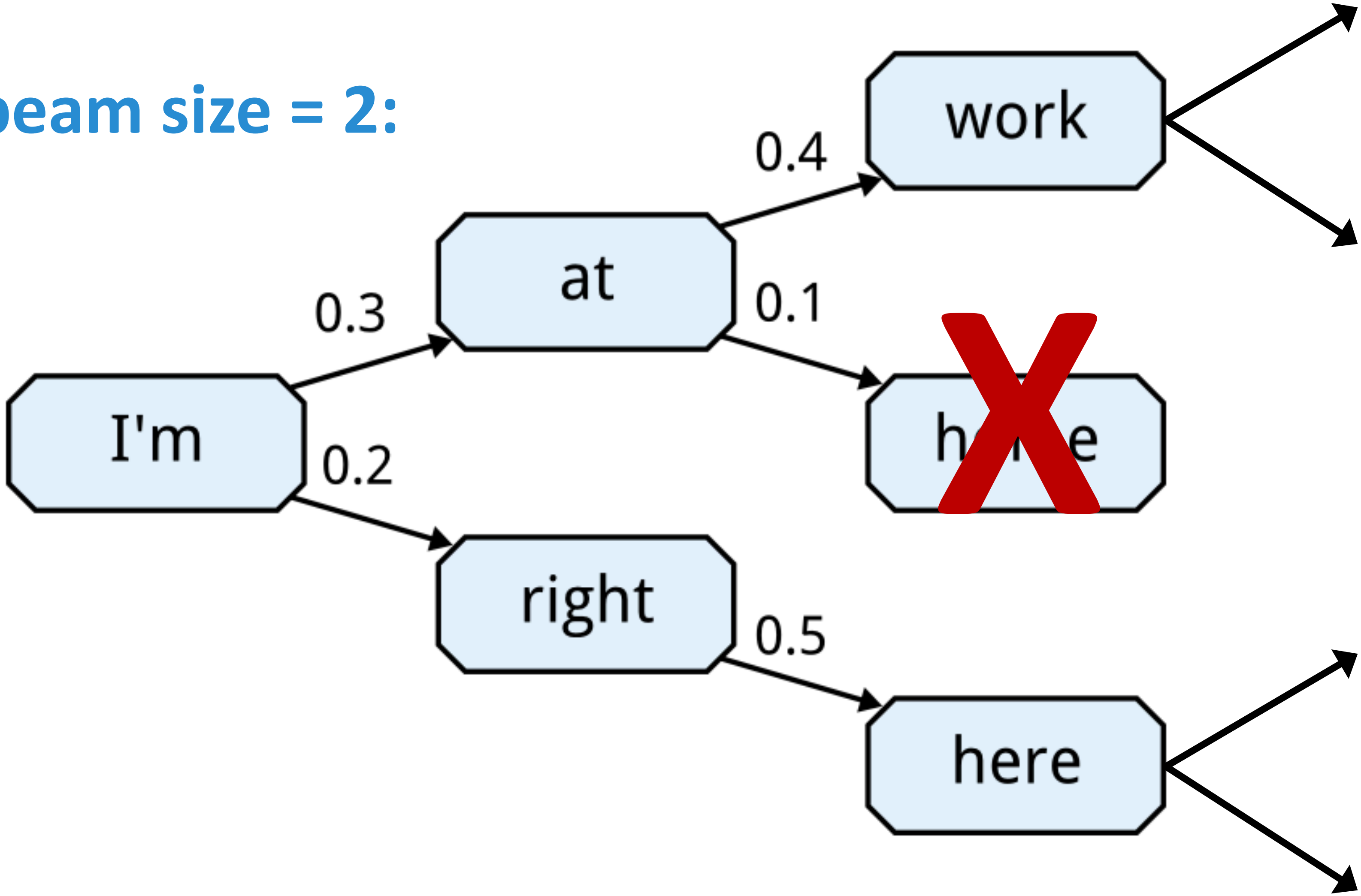
- beam search alternates between extending hypotheses and pruning hypothesis sets
- the design of these steps depends on the structure being predicted
- at the end, just return the highest-scoring hypothesis
- the final set of hypotheses can also be used as an approximate n-best list (where $n = b$)

Beam Search

- if we set $b = |L|$, do we get Viterbi?
  - no
  - beam search still operates left-to-right greedily and can't recover if the best path is pruned early
  - Viterbi doesn't prune
- **recombination** can improve the diversity of hypotheses in the beam (and therefore improve the search), but is only applicable for certain parts functions

Beam Search for Generation

Let beam size = 2:



Pranav Khaitan, Google Research Blog: *Chat Smarter with Allo*

Beam Search in Generation

- in generation tasks, using too large of a beam size may hurt performance
- why?

Approximate Inference

- greedy
- beam search
- coarse-to-fine
- heuristic search

Coarse-to-Fine

- use a series of models of increasing complexity
  - earlier models are faster than later models
  - each model is used to prune away potential structures for subsequent models to consider
- downside is that this requires training additional models
  - but these additional models are usually fairly simple and efficient to train

Coarse-to-Fine

- this is popular for tasks like parsing

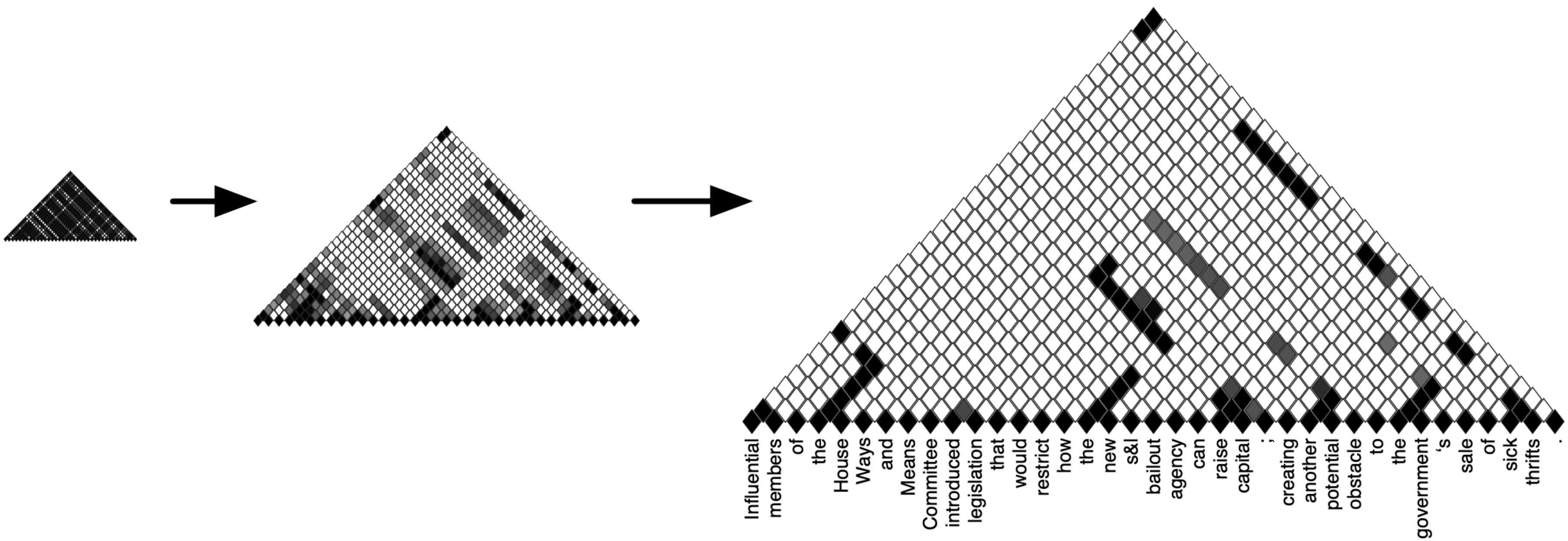


Figure 1.3. Charts are used to depict the dynamic programming states in parsing. In coarse-to-fine parsing, the sentence is repeatedly re-parsed with increasingly refined grammars, pruning away low probability constituents. Finer grammars need to only consider only a fraction of the enlarged search space (the non-white chart items).

Petrov (2009): *Coarse-to-Fine Natural Language Processing*

Coarse-to-Fine

- also can be used for generation tasks (by clustering words and training coarse models to predict clusters)

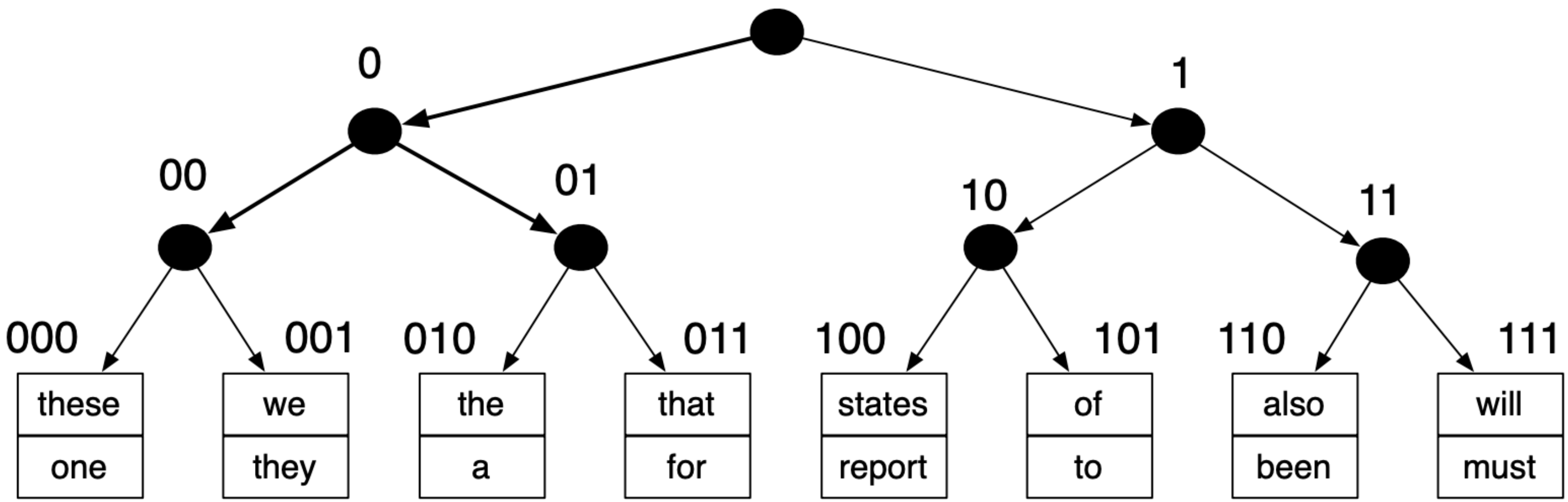


Figure 5.1. An example of hierarchical clustering of target language vocabulary (see Section 5.4). Even with a small number of clusters our divisive HMM clustering (Section 5.4.3) captures sensible syntactico-semantic classes.

Petrov (2009): *Coarse-to-Fine Natural Language Processing*

Coarse-to-Fine

- remember the local predictors we discussed for dependency parsing and machine translation?
- while they don't work very well by themselves, they can be useful as coarse models
- e.g., for dependency parsing:
  - train a local predictor
  - use it to get top *k* head candidates for each word
  - restrict next model to trees that use those candidates

References for Coarse-to-Fine Procedures in NLP

- Petrov (2009): *Coarse-to-Fine Natural Language Processing*
- Weiss and Taskar (2010): *Structured Prediction Cascades*
- Rush and Petrov (2012): *Vine Pruning for Efficient Multi-Pass Dependency Parsing*

Heuristic Search Algorithms

- beam search can be improved by using heuristics to favor certain hypothesis extensions over others
  - e.g., in phrase-based machine translation this is called "future cost estimation" (see Koehn et al. (2003): *Statistical Phrase-Based Translation)*

- if using a particular form of beam search (cf. "agenda algorithms") and the heuristics satisfy certain conditions, search can be exact
  - cf. A* search
  - for parsing, see Klein & Manning (2003): *A* Parsing: Fast Exact Viterbi Parse Selection*

Non-Local Features

- efficient exact or even approximate inference requires relatively small parts
- but intuitively, this limits modeling power
- how can we combine efficiency with some long-distance or “non-local” information in the scoring function?
- lots of work on this

Non-Local Features in Named Entity Recognition

… But in the end, Chicago came up short.

organization?
location?

Non-Local Features in Named Entity Recognition

organization

- The Chicago Bears needed a win in Sunday night's game.... But in the end, Chicago came up short.

organization

- first mention of a named entity may have more information

Non-Local Features in Named Entity Recognition

- this type of non-local feature was used in several papers focused on approximate inference for NLP

Skip-Chain CRFs with Inference via Loopy Belief Propagation

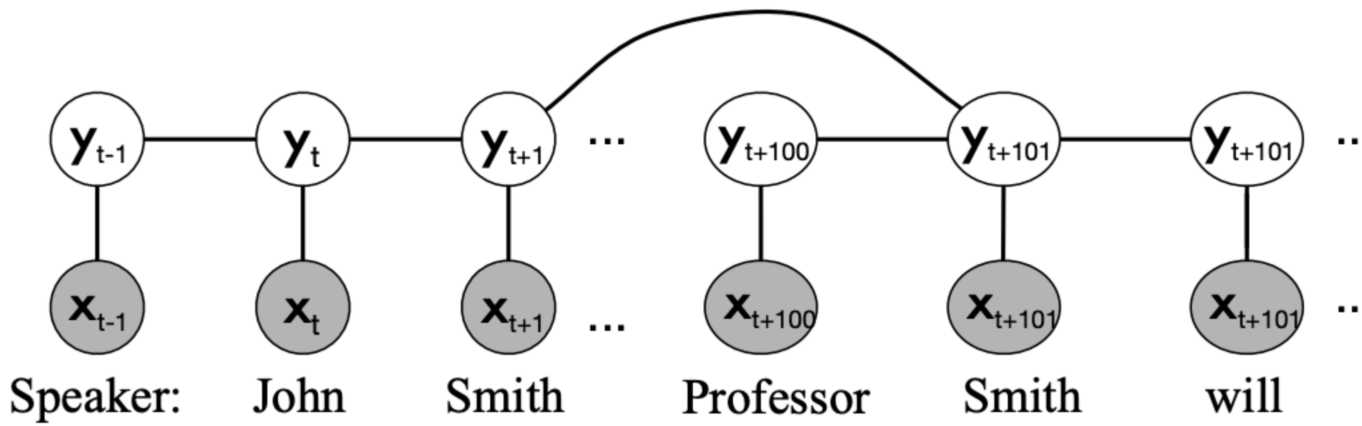


Figure 2: Graphical representation of skip-chain CRF. Identical words are connected because they are likely to have the same label.

Sutton and McCallum (2004): *Collective Segmentation and Labeling of Distant Entities in Information Extraction*

Inference via Gibbs Sampling

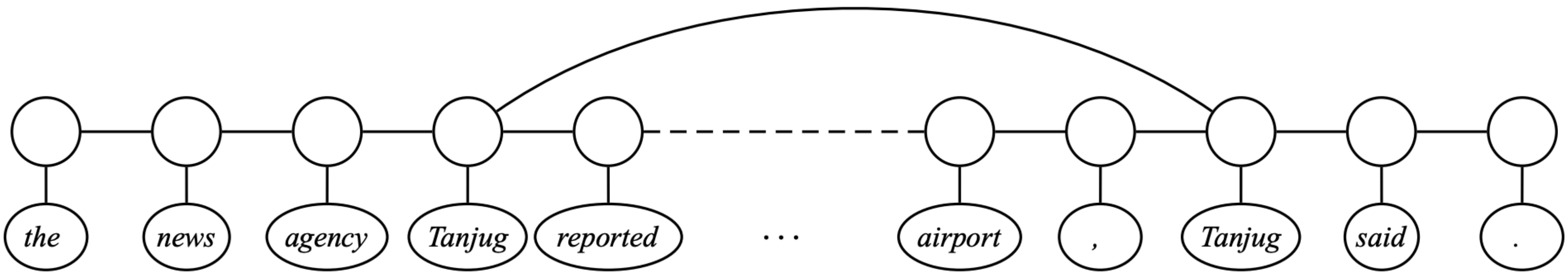
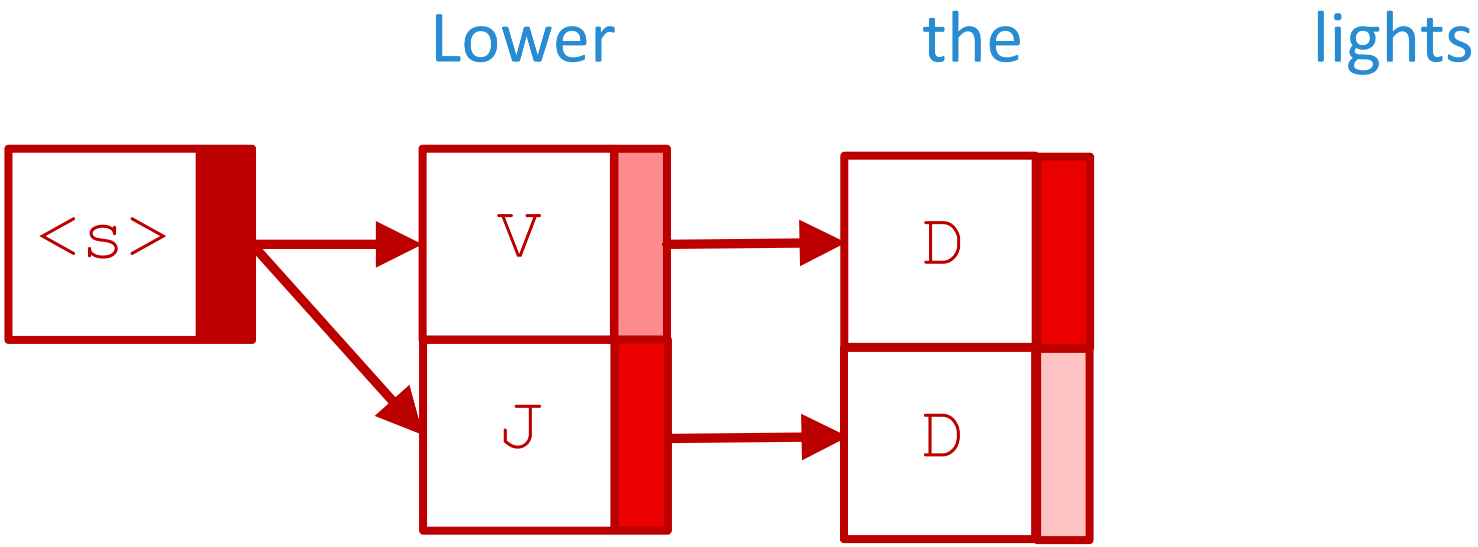


Figure 1: An example of the label consistency problem excerpted from a document in the CoNLL 2003 English dataset.

Finkel et al. (2005): *Incorporating non-local information into information extraction systems by Gibbs sampling*

Non-Local Features in Beam Search

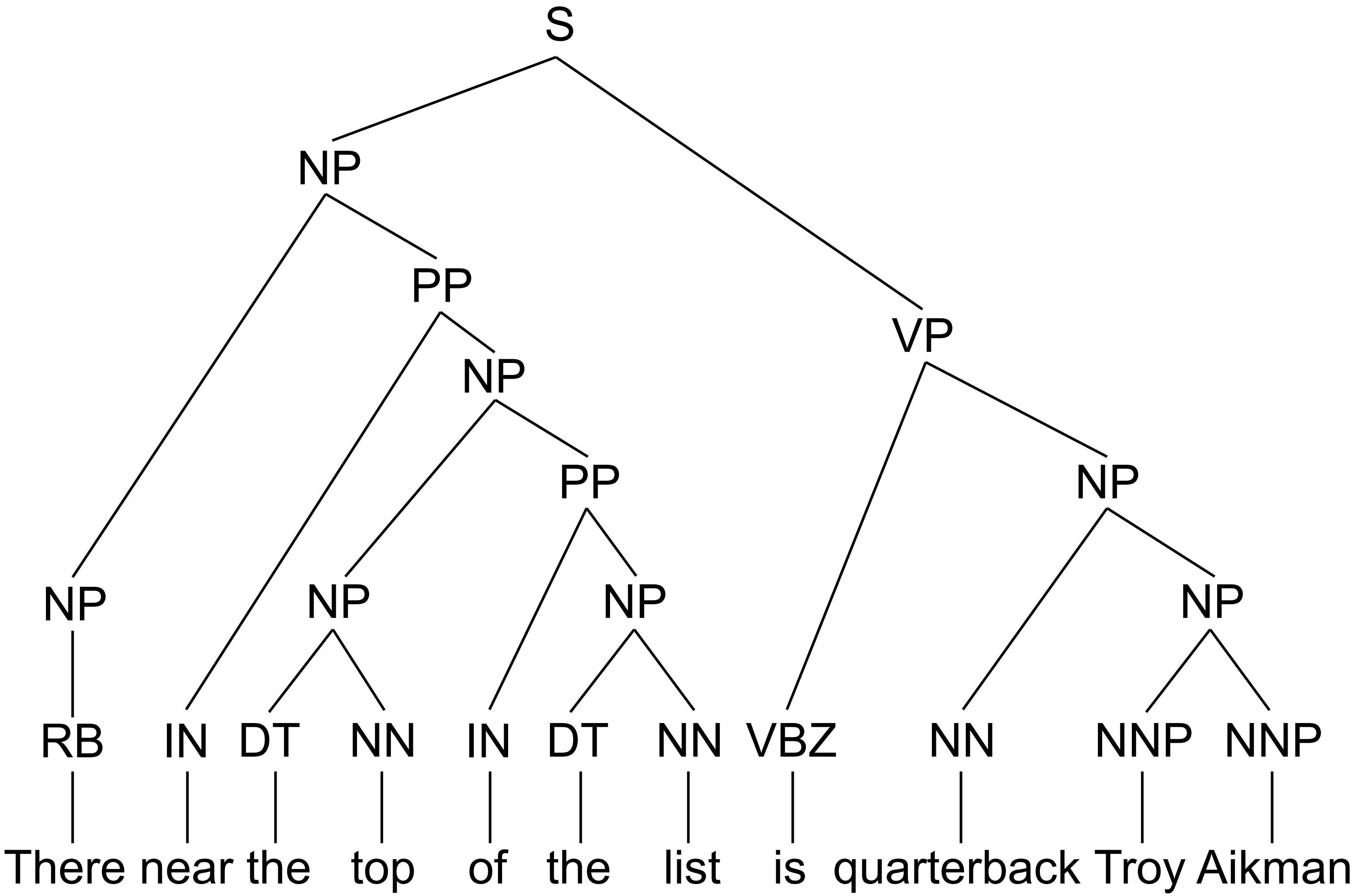


note: using backpointers, we can recover the entire hypothesis → we can compute any feature or scoring function using the entire hypothesis!

same idea can be applied to Viterbi and other exact DP algorithms!

Inference in PCFGs

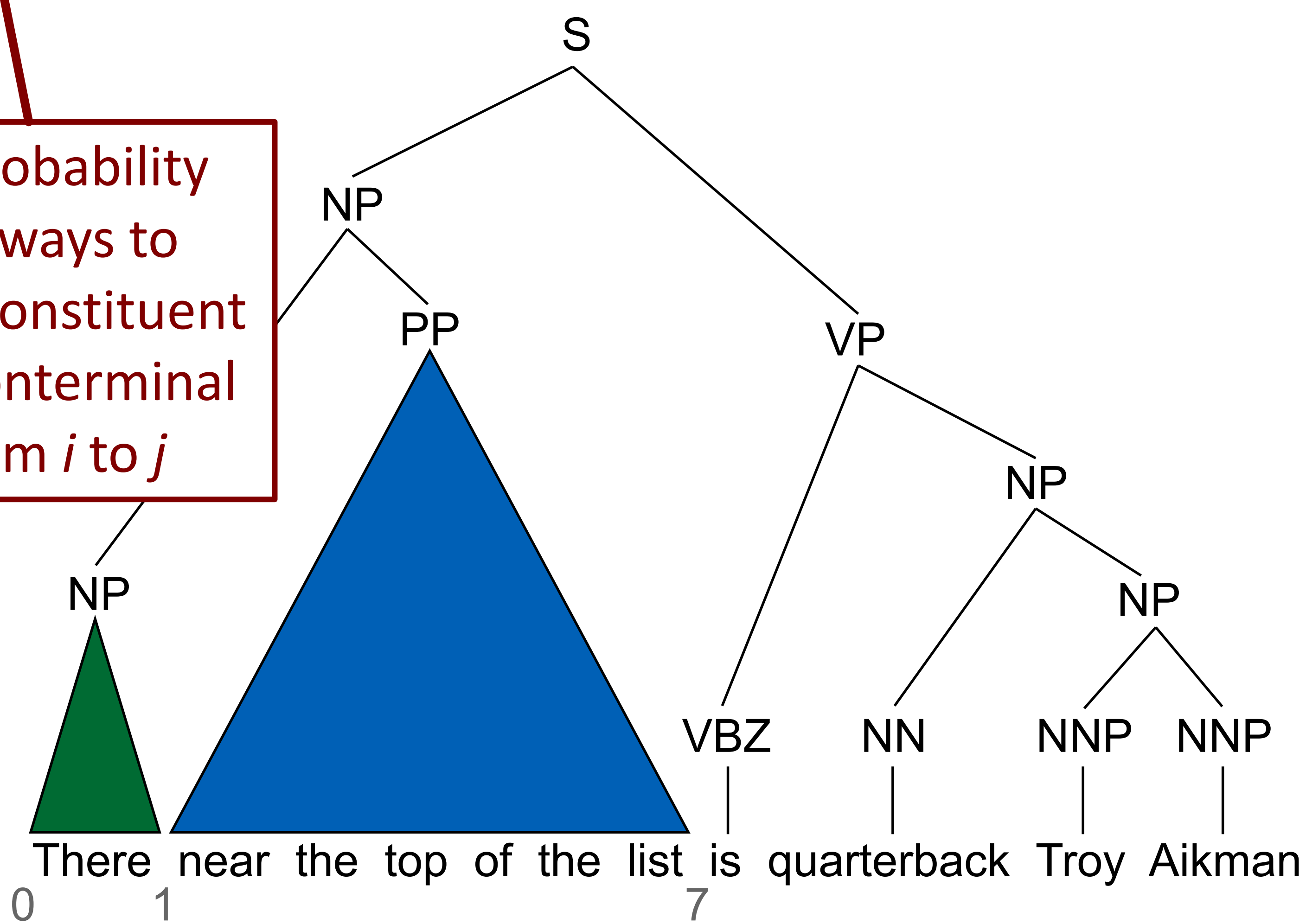
- to find max-probability tree for a sentence, use dynamic programming: **CKY algorithm**
- to find the best way to build a tree covering words *i* to *j*:
  - consider all possible "split points" *k* between *i* and *j*
  - for each split point *k*, consider all possible nonterminals for the two smaller trees created by that split


S
NP
VP
PP
NP
NP
PP
NP
NP
NP
NP
RB IN DT NN IN DT NN VBZ NN NNP NNP
There near the top of the list is quarterback Troy Aikman

CKY Algorithm

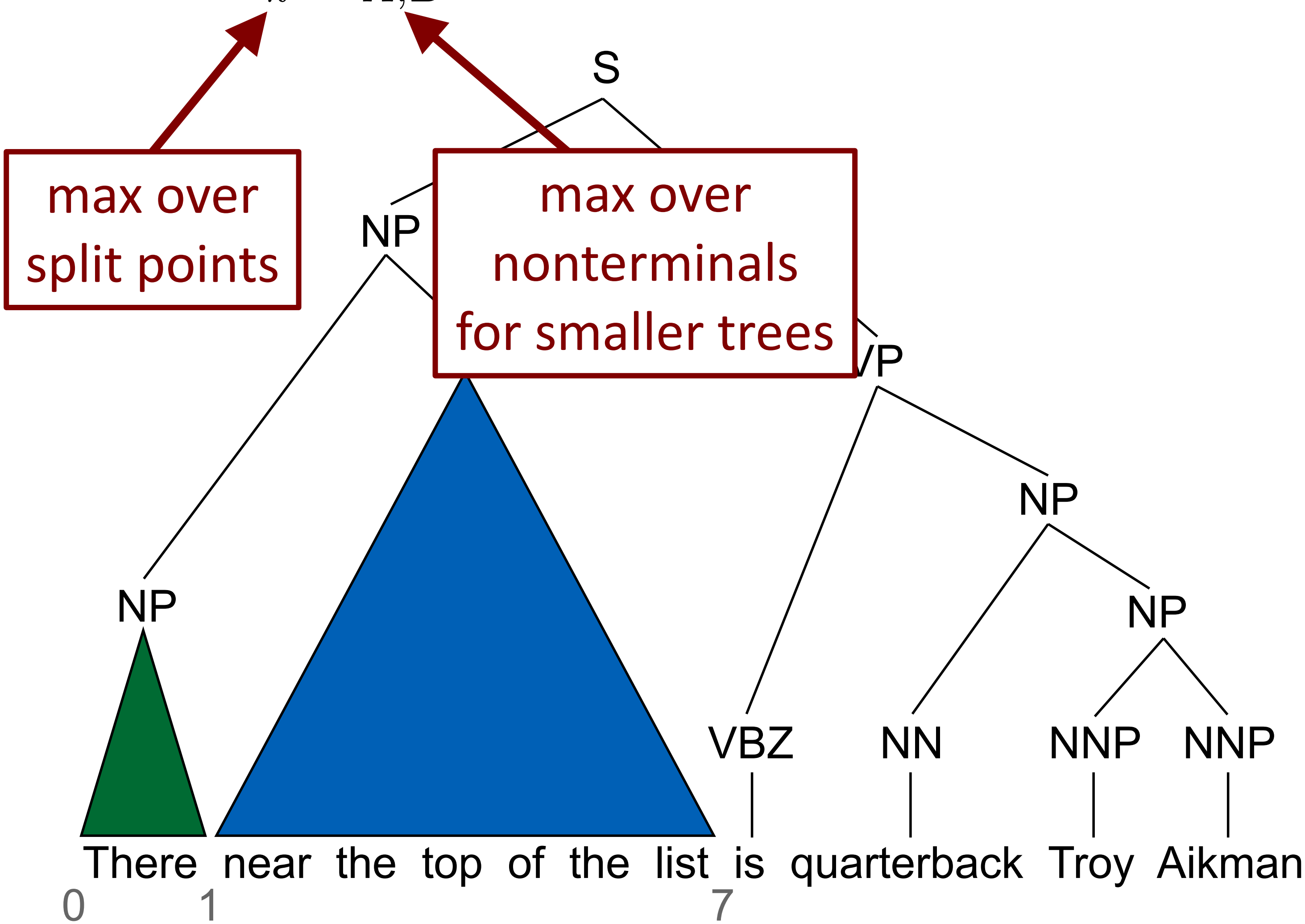
$$C(Z,i,j) = \max_{k} \max_{A,B} \left(C(A,i,k)\, C(B,k,j)\, \mathrm{score}(\langle Z \to A\, B \rangle)\right)$$

max probability of all ways to build a constituent with nonterminal *Z* from *i* to *j*



CKY Algorithm

$$C(Z,i,j) = \max_{k} \max_{A,B} \left(C(A,i,k)\, C(B,k,j)\, \text{score}(\langle Z \to A\, B \rangle) \right)$$

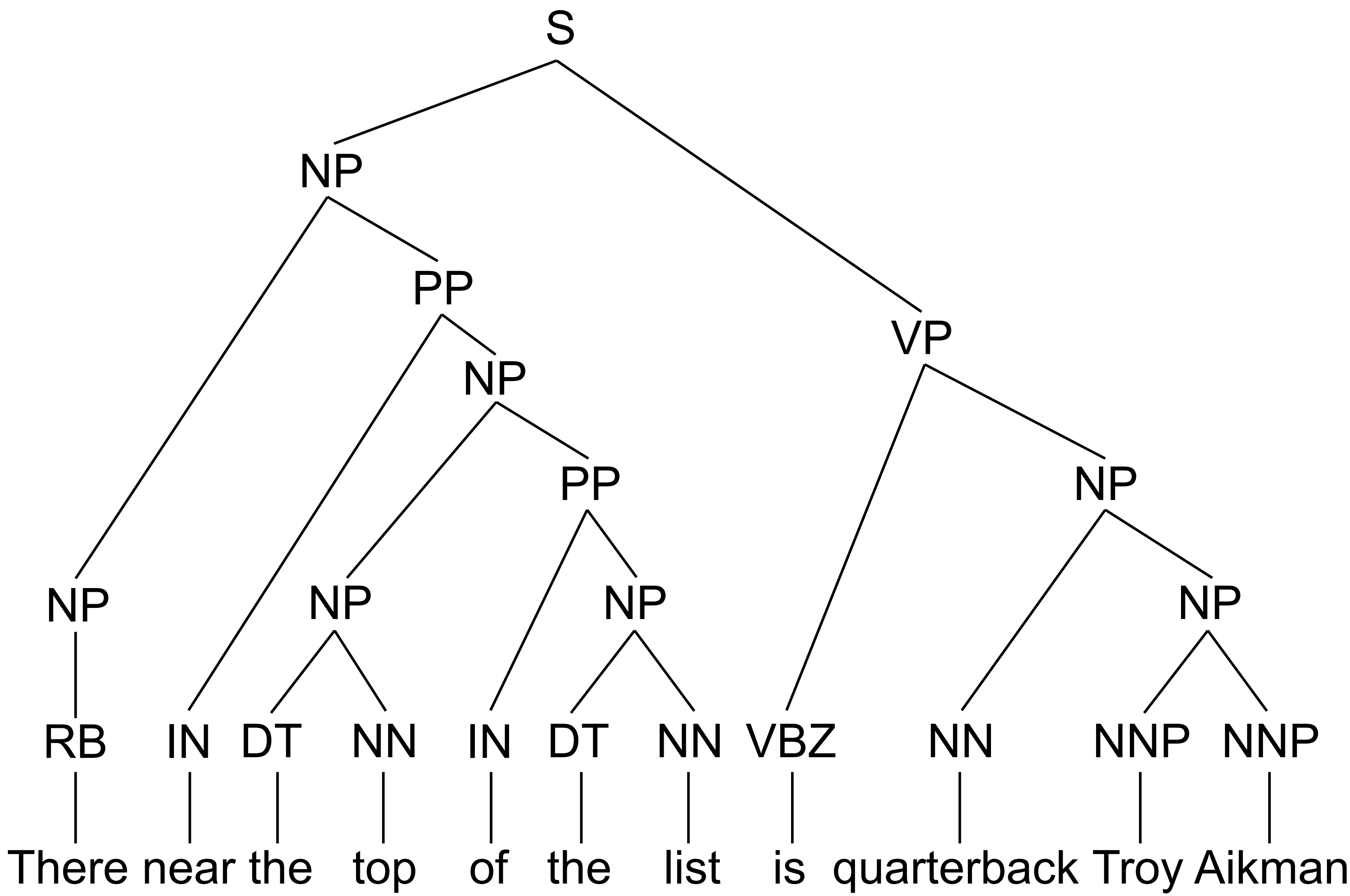


- detail: CKY requires the PCFG to be in **Chomsky Normal Form (CNF)**
- basically: every rule has either 2 nonterminals or 1 terminal on the right-hand side

Cube Pruning

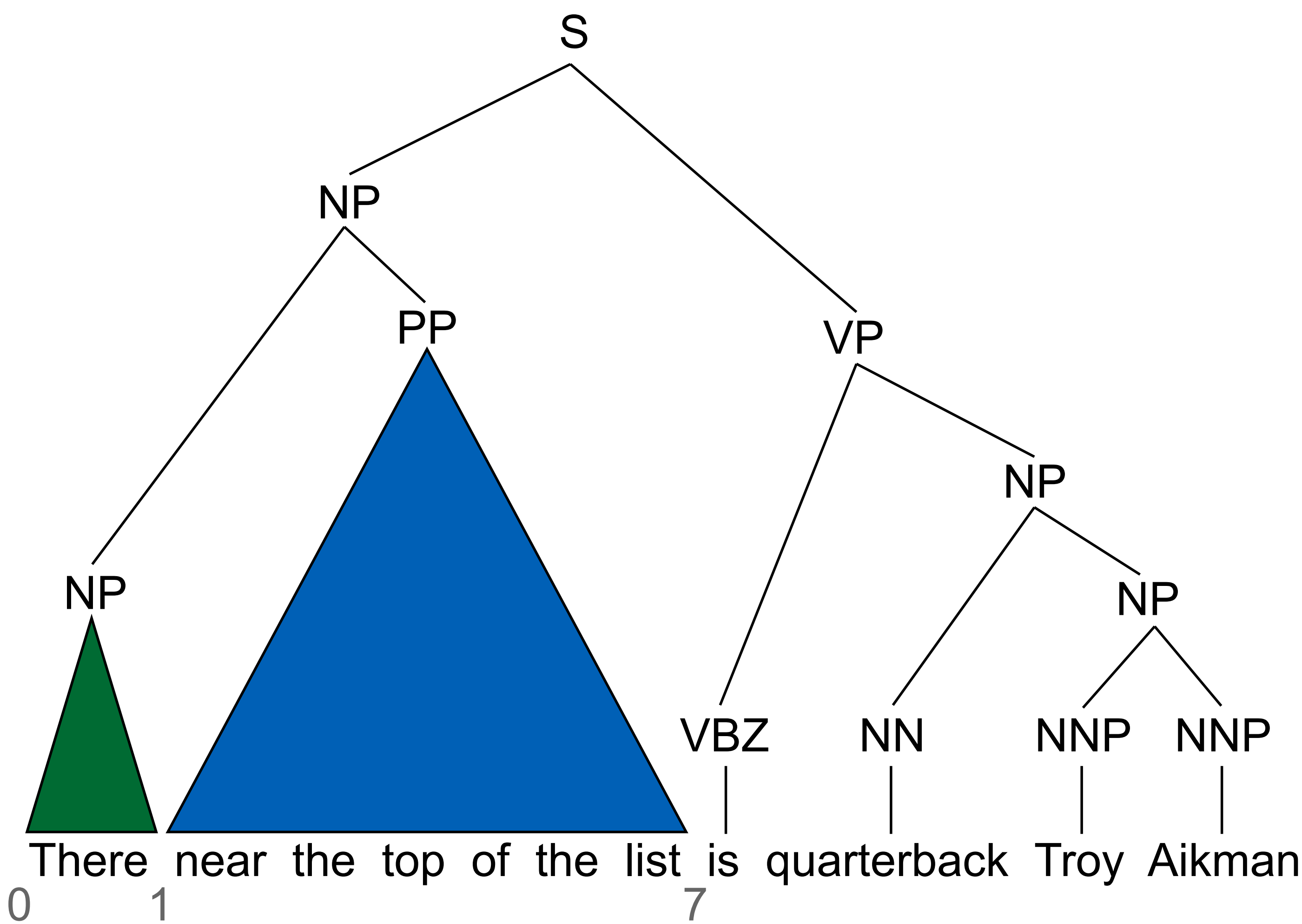
(Chiang, 2007; Huang & Chiang, 2007)

- modification to dynamic programming algorithms for decoding to use non-local features approximately
- keeps a k-best list of derivations for each item
- applies non-local feature functions on these derivations when defining new items
- lets modeler decide which scoring terms to incorporate exactly and which scoring terms to incorporate approximately

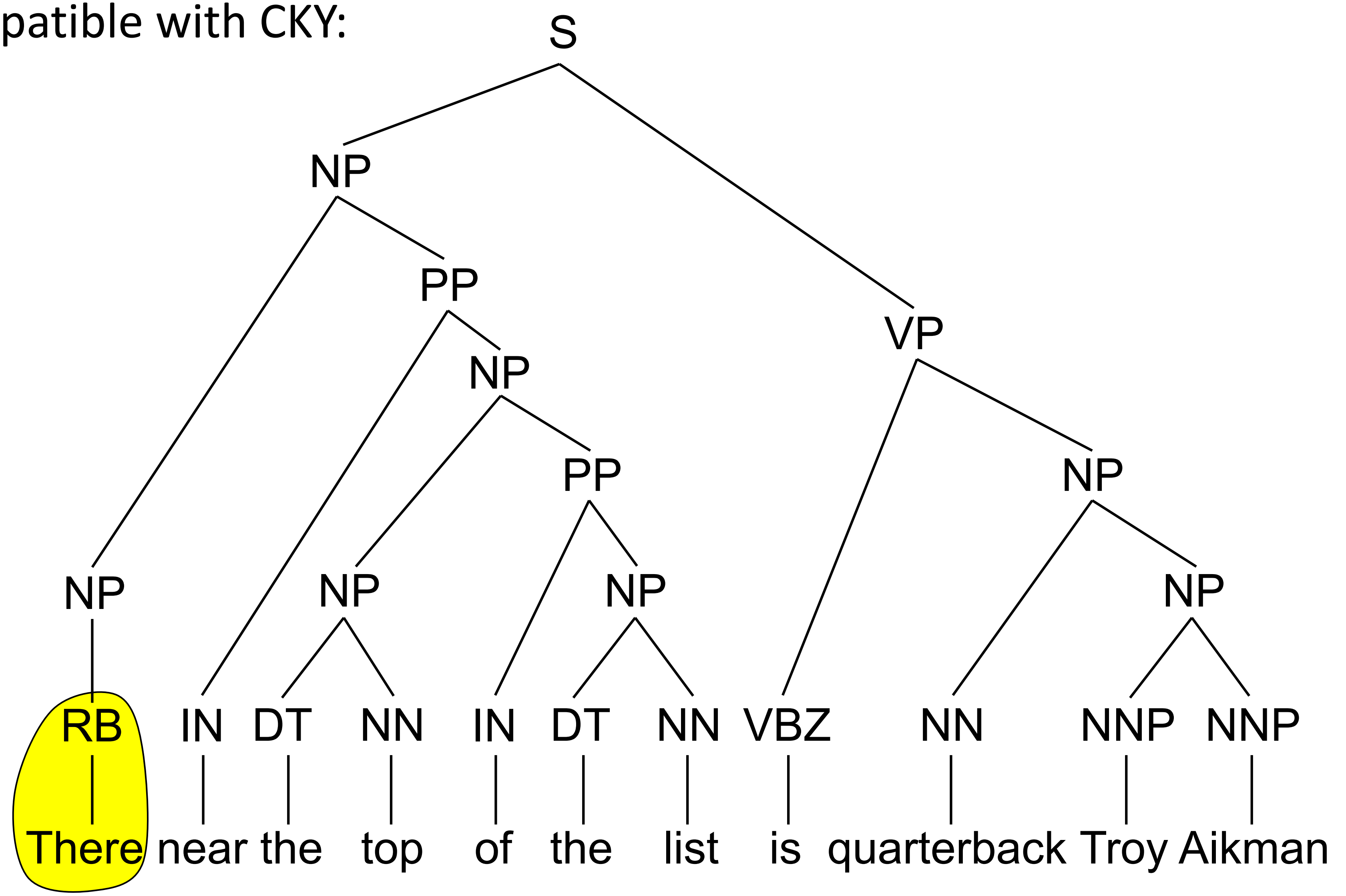

S
NP
VP
PP
NP
NP
PP
NP
NP
NP
NP
RB IN DT NN IN DT NN VBZ NN NNP NNP
There near the top of the list is quarterback Troy Aikman

CKY Algorithm

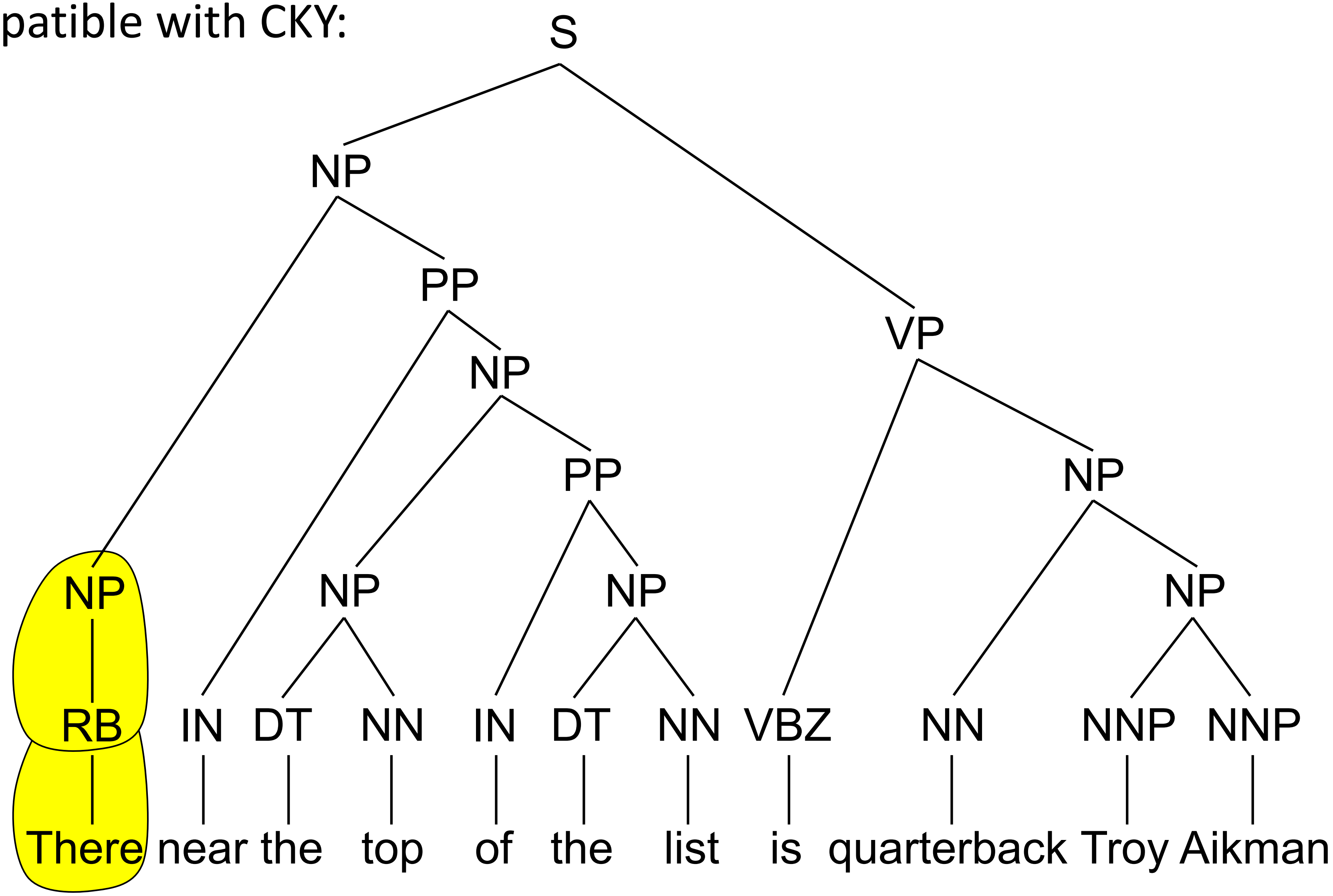
$$C(Z,i,j) = \max_{k} \max_{A,B} \left(C(A,i,k)\, C(B,k,j)\, \mathrm{score}(\langle Z \to A\, B \rangle) \right)$$



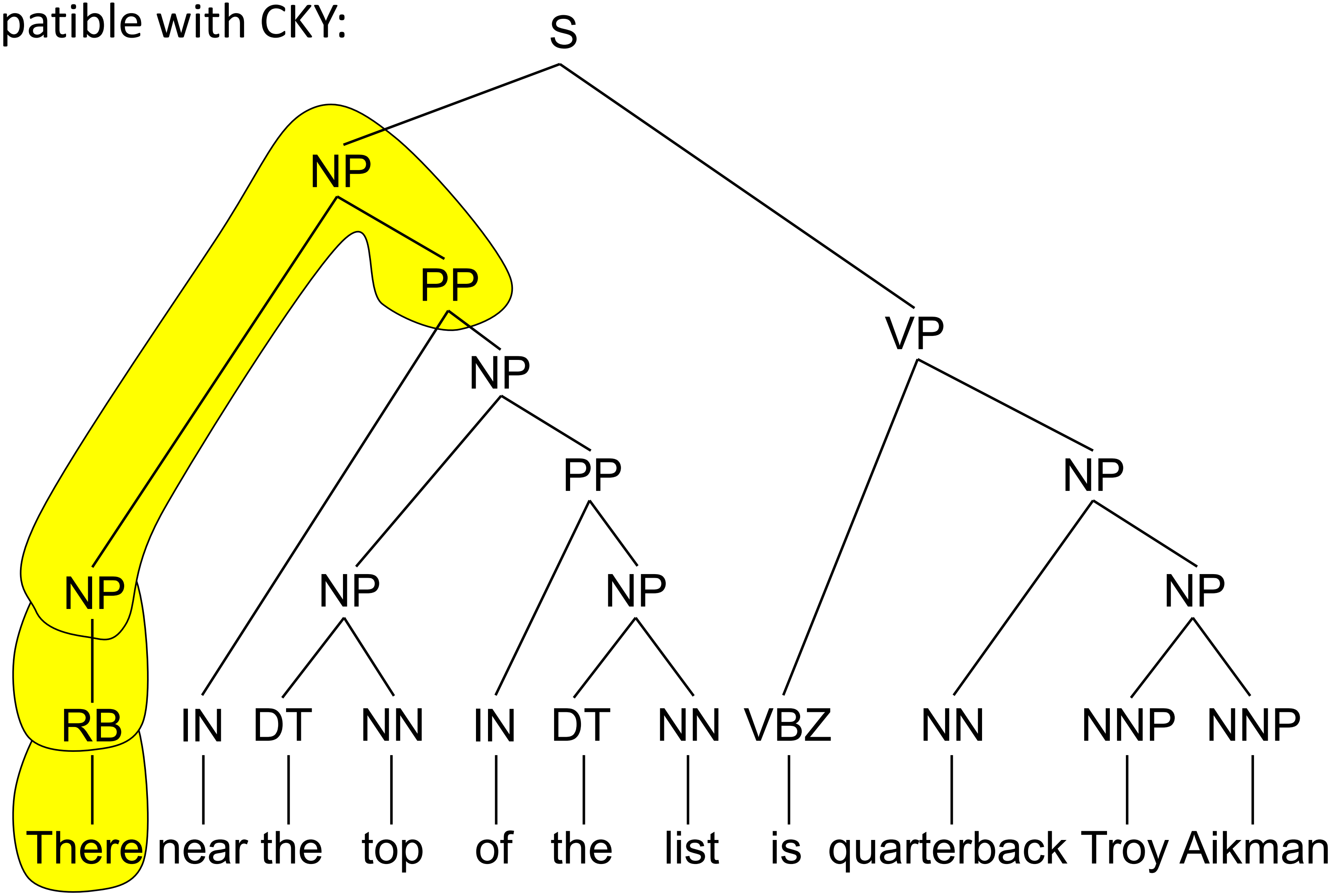
Local scoring functions compatible with CKY:



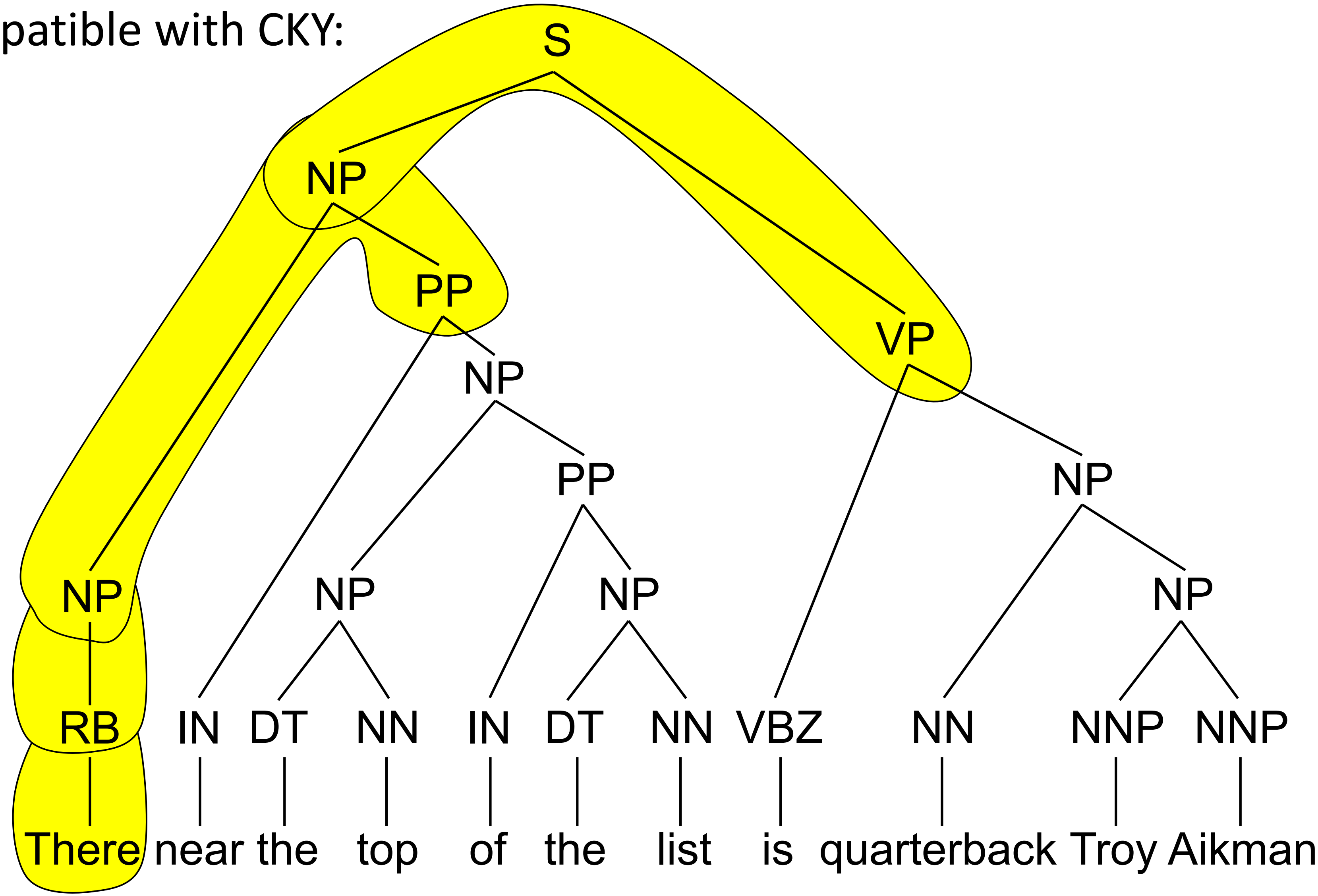
Local scoring functions compatible with CKY:



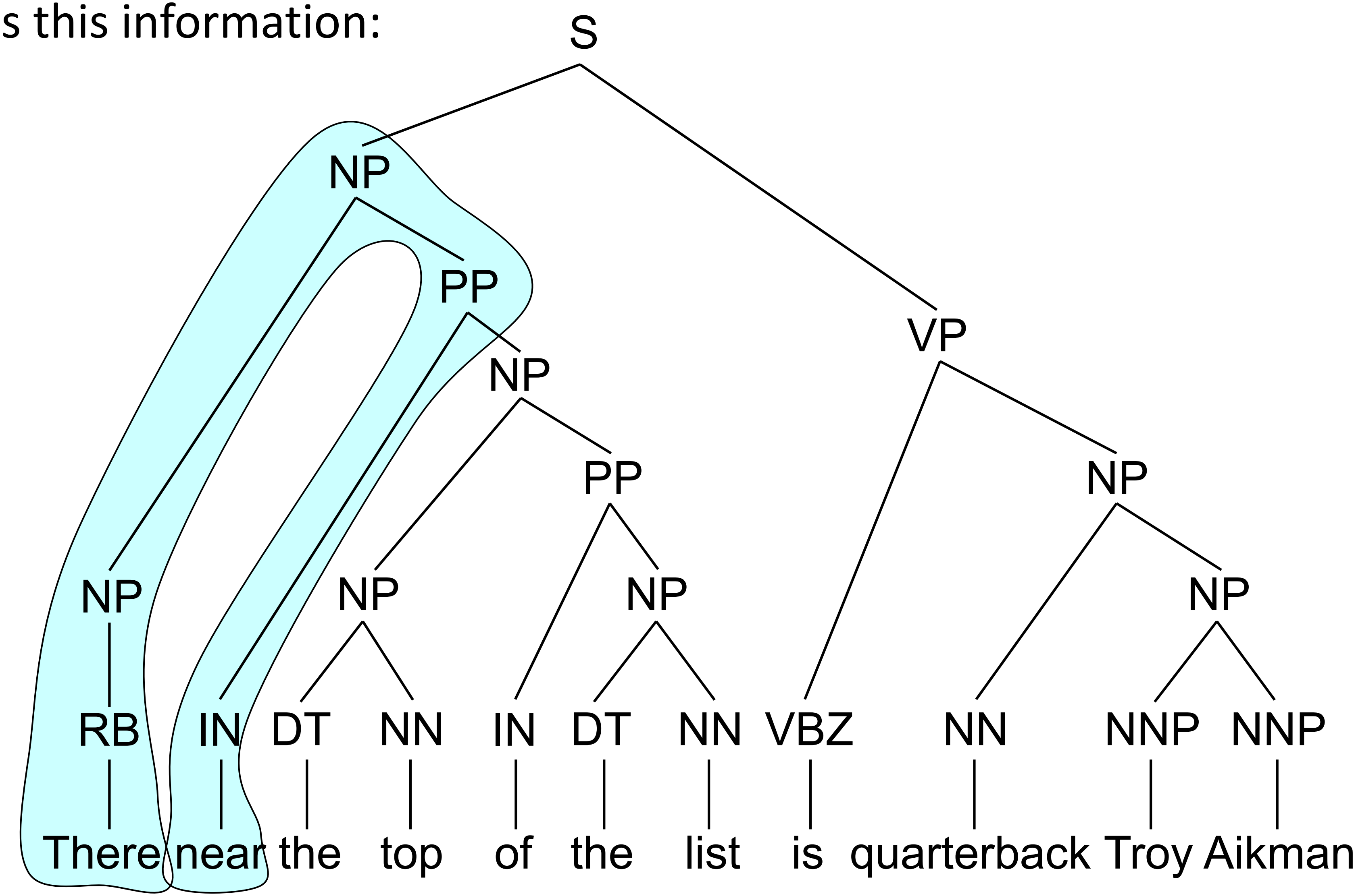
Local scoring functions compatible with CKY:



Local scoring functions compatible with CKY:

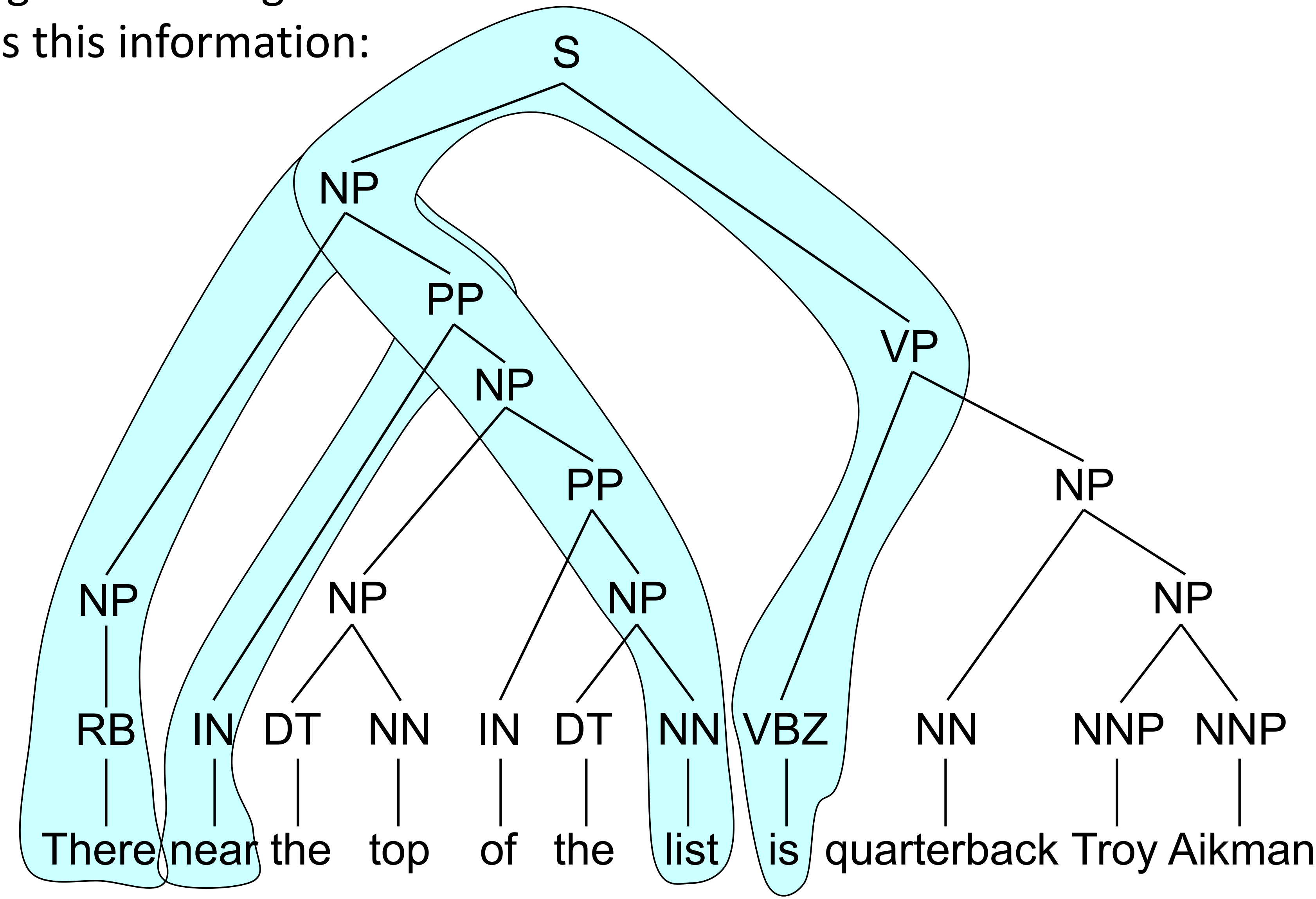


Imagine a scoring term that uses this information:



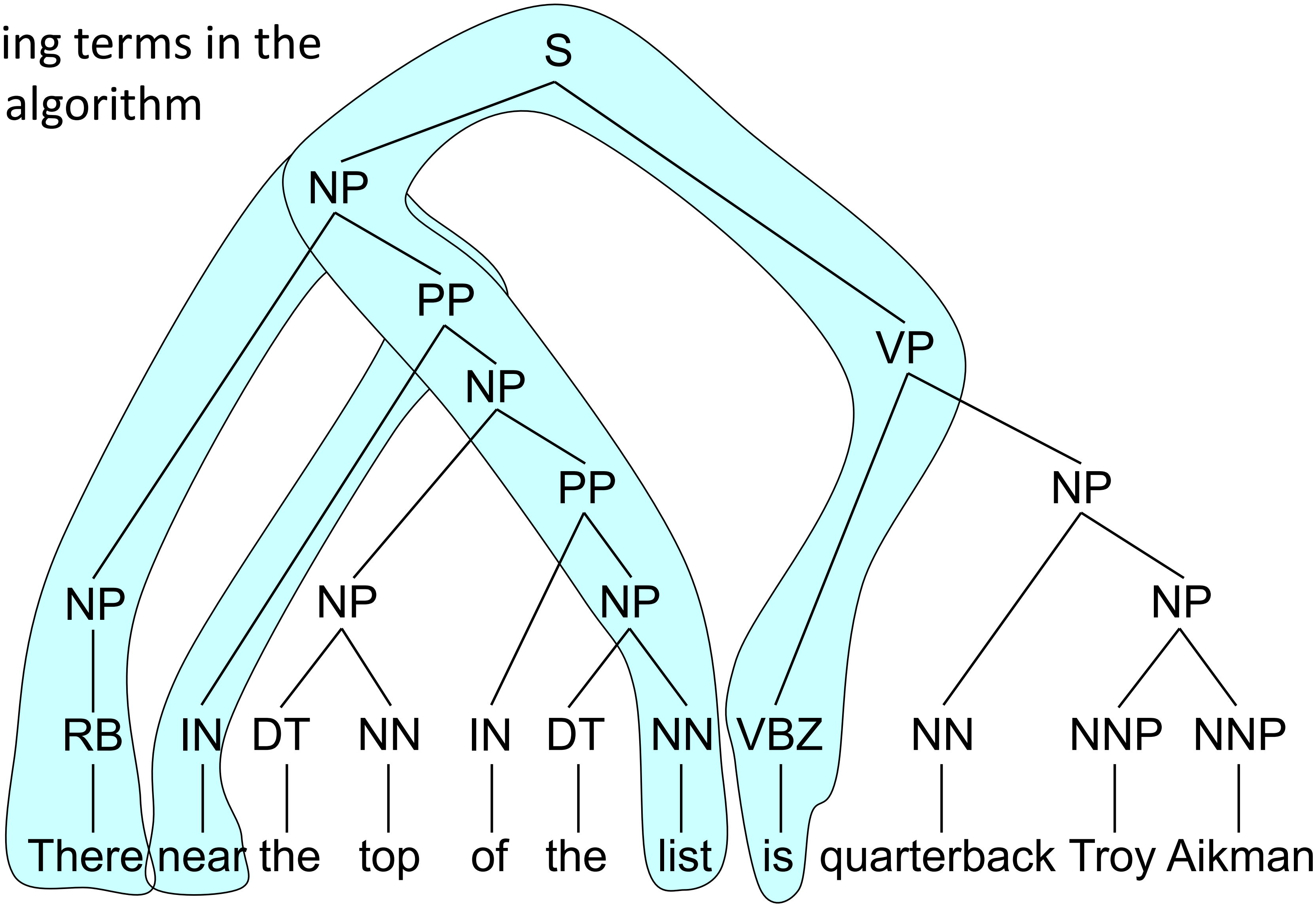
“NGramTree” feature
(Charniak & Johnson, 2005)

Imagine a scoring term that uses this information:



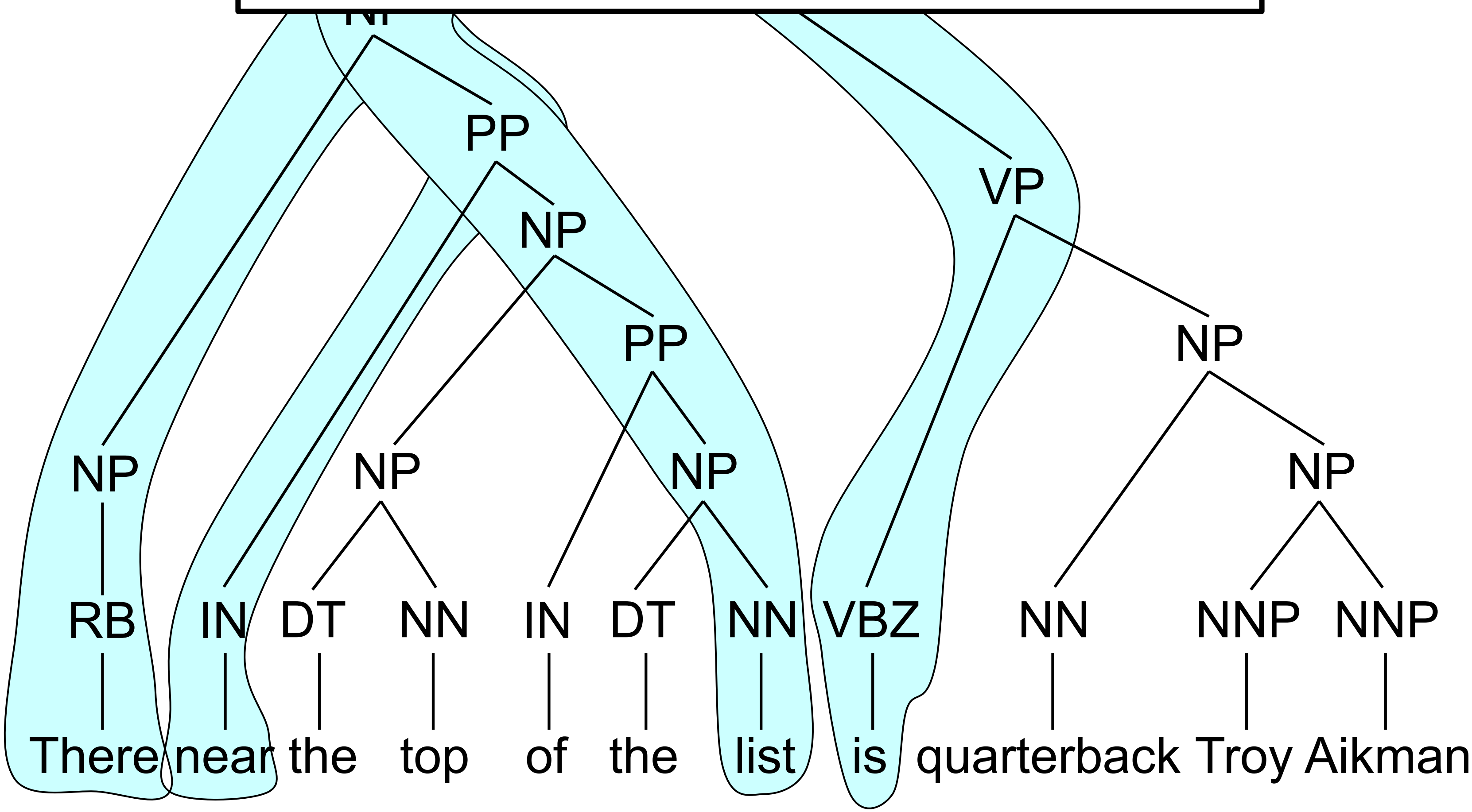
“NGramTree” feature
(Charniak & Johnson, 2005)

We cannot use these scoring terms in the CKY algorithm

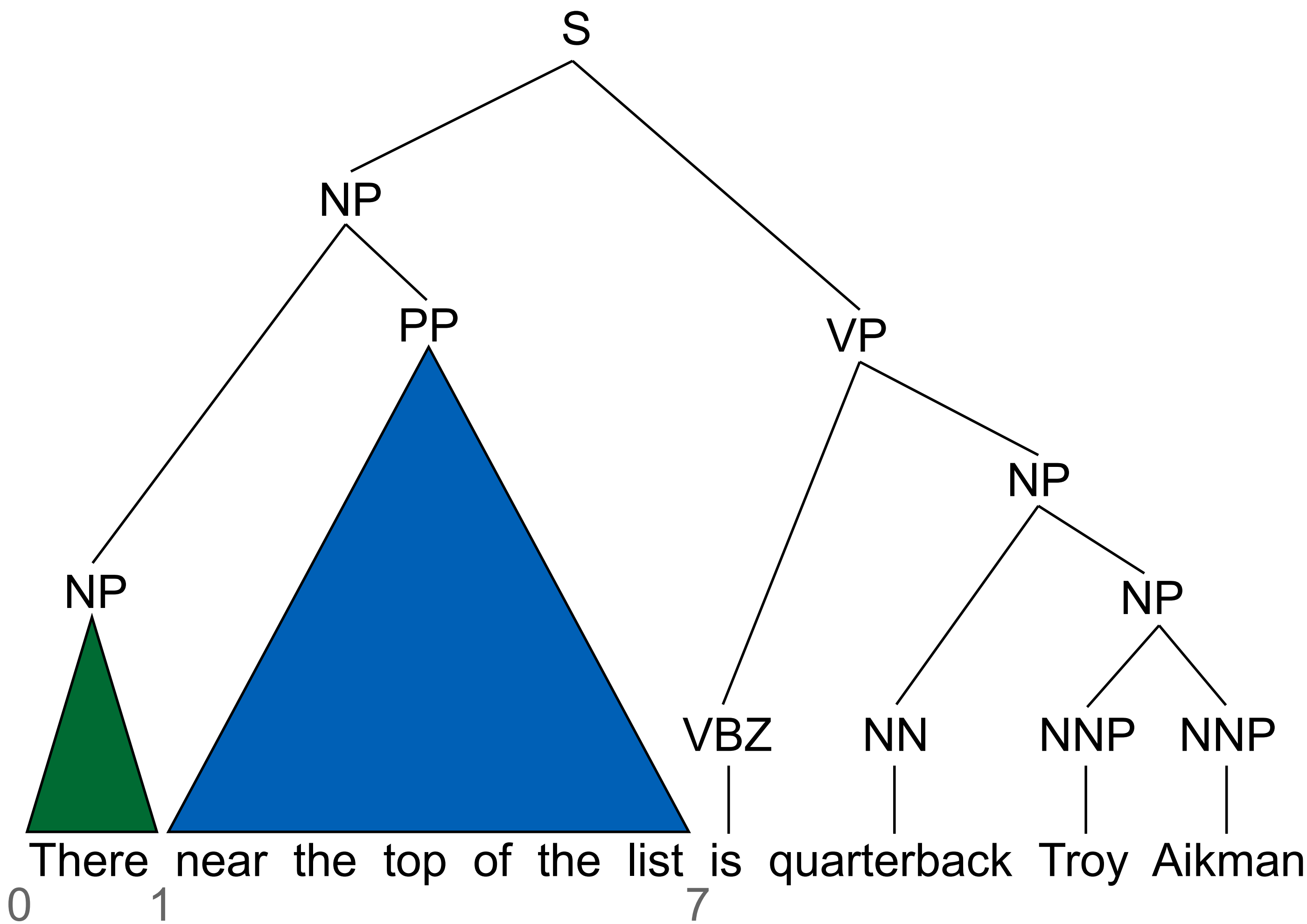


$$C(Z, i, j) = \max_{k} \max_{A,B} \left(C(A, i, k)\, C(B, k, j)\, \mathrm{score}(\langle Z \to A\, B \rangle)\right)$$

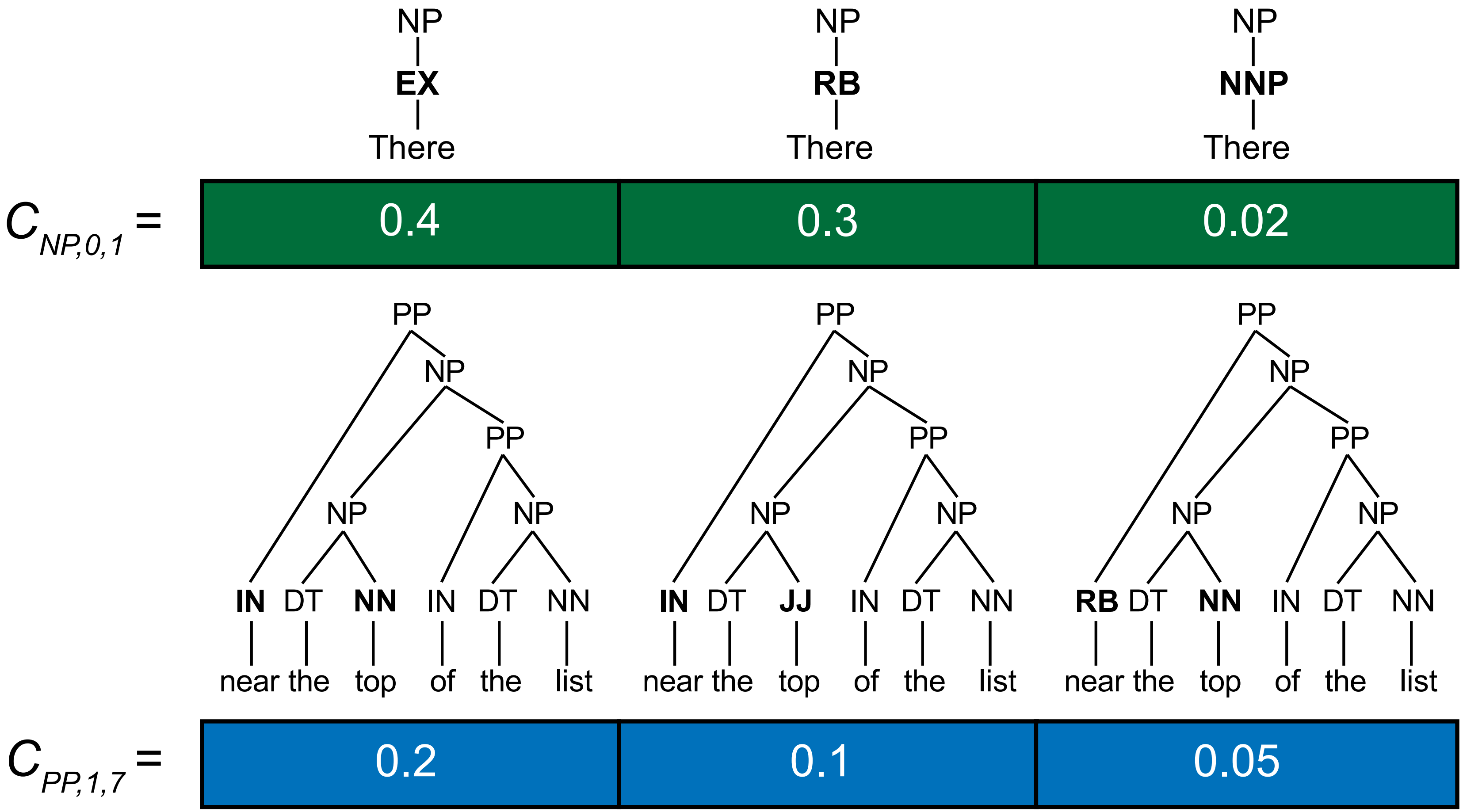
“non-local features” like these break dynamic programming!



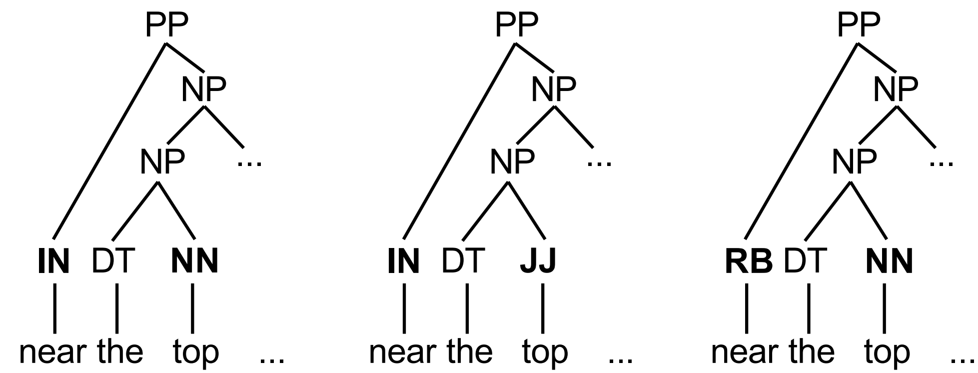
$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \to NP\ PP}$$

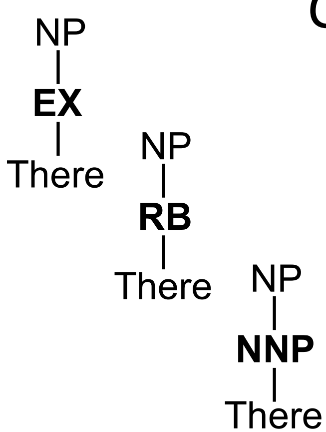


$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$



$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

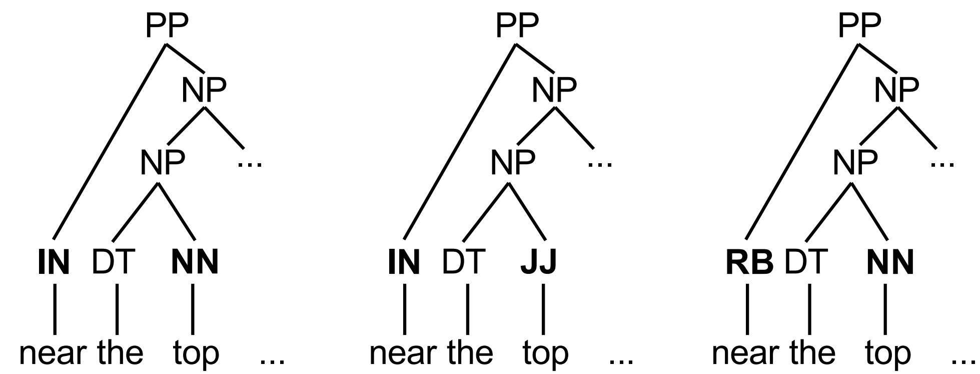




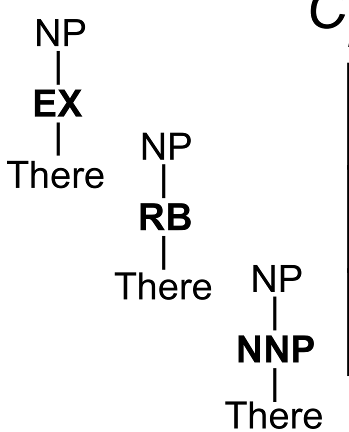
$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.08	0.04	0.02
0.3	0.06	0.03	0.015
0.02	0.004	0.002	0.001

$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

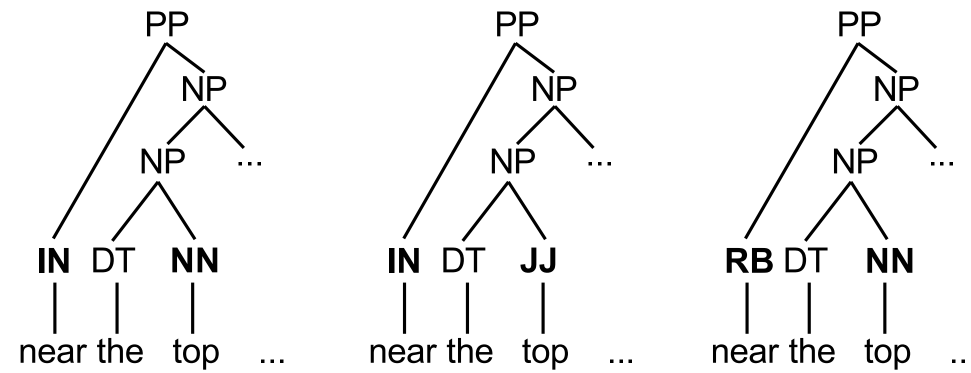
$$\lambda_{NP \rightarrow NP\ PP} = 0.5$$

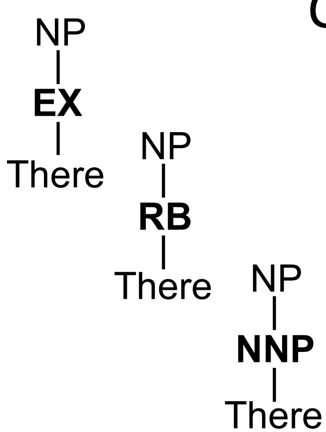


$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.08 × 0.5	0.04 × 0.5	0.02 × 0.5
0.3	0.06 × 0.5	0.03 × 0.5	0.015 × 0.5
0.02	0.004 × 0.5	0.002 × 0.5	0.001 × 0.5



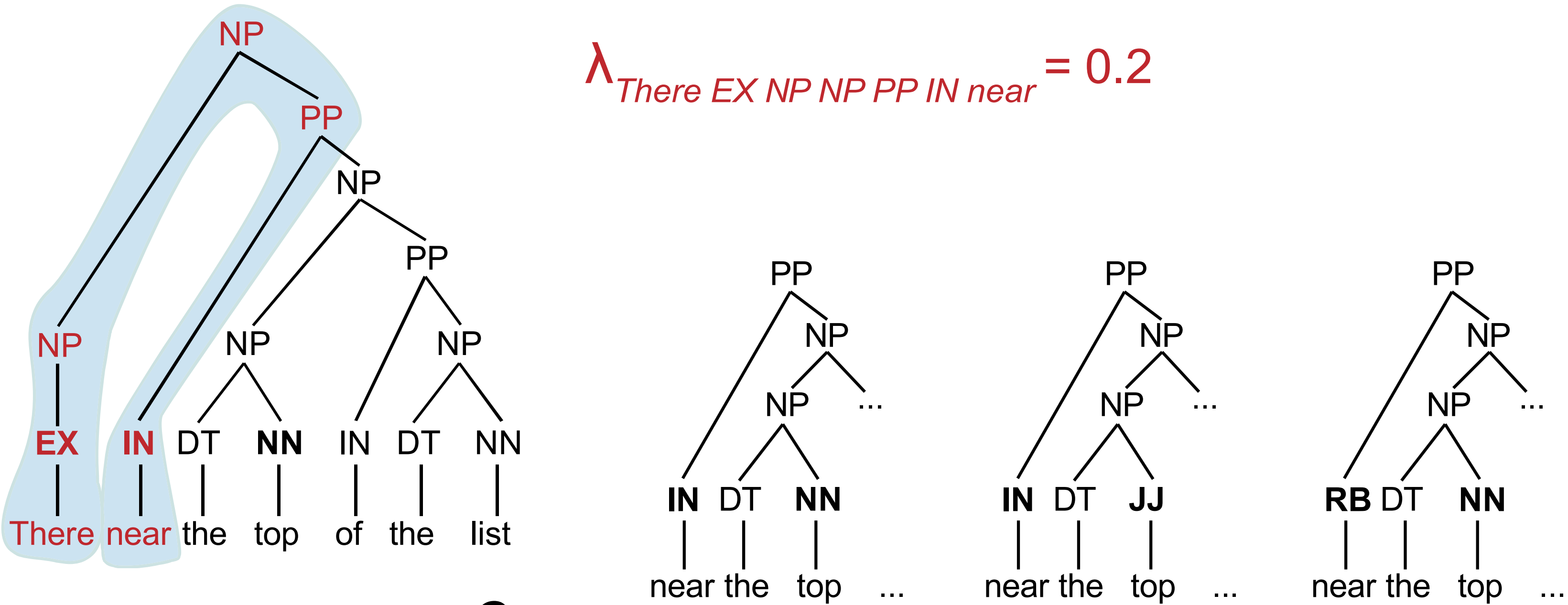
$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

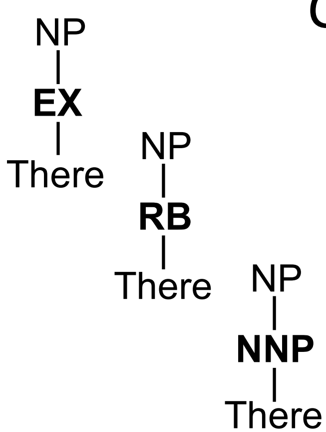




$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04	0.02	0.01
0.3	0.03	0.015	0.0075
0.02	0.002	0.001	0.0005

$\lambda_{There\ EX\ NP\ NP\ PP\ IN\ near} = 0.2$





$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04 × 0.2	0.02 × 0.2	0.01
0.3	0.03	0.015	0.0075
0.02	0.002	0.001	0.0005

$\lambda_{There\ EX\ NP\ NP\ PP\ IN\ near} = 0.2$

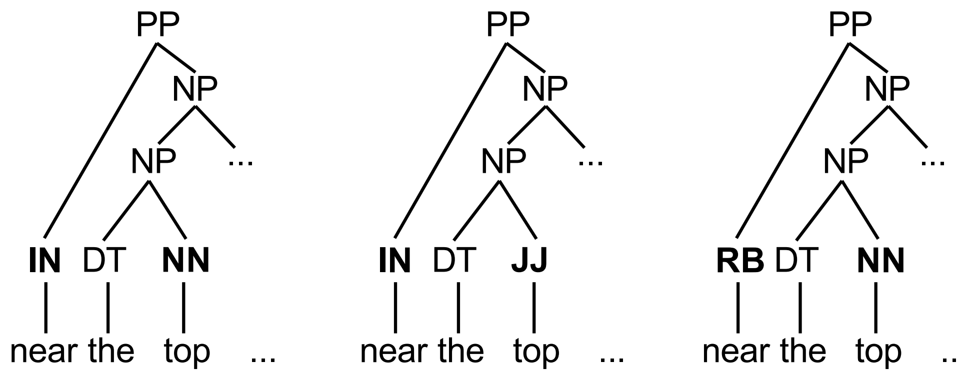
$\lambda_{There\ RB\ NP\ NP\ PP\ IN\ near} = 0.6$

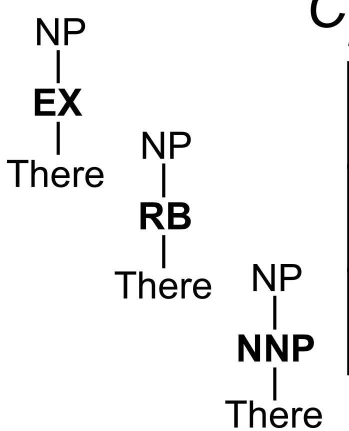
$\lambda_{There\ NNP\ NP\ NP\ PP\ IN\ near} = 0.1$

$\lambda_{There\ EX\ NP\ NP\ PP\ RB\ near} = 0.1$

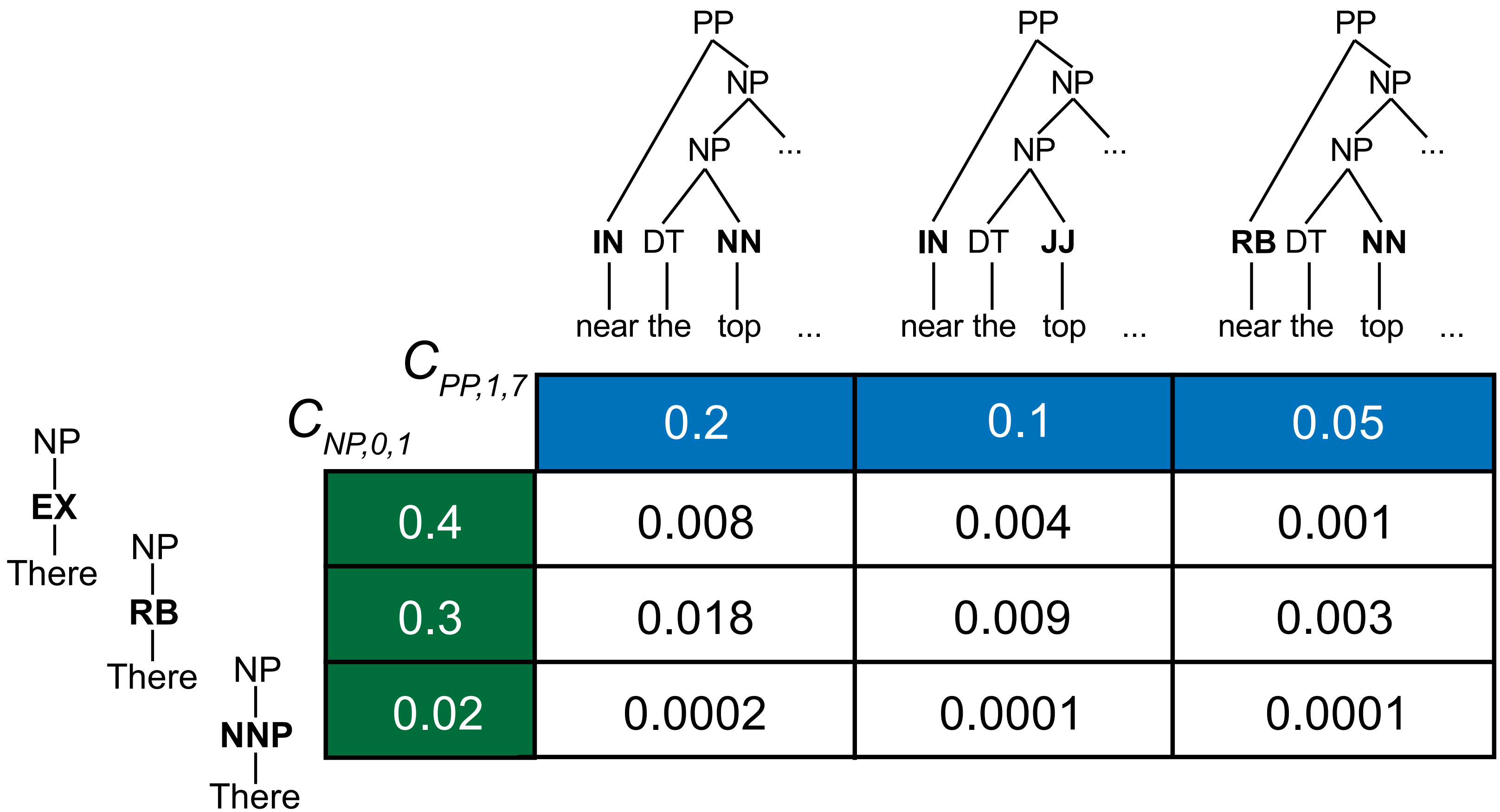
$\lambda_{There\ RB\ NP\ NP\ PP\ RB\ near} = 0.4$

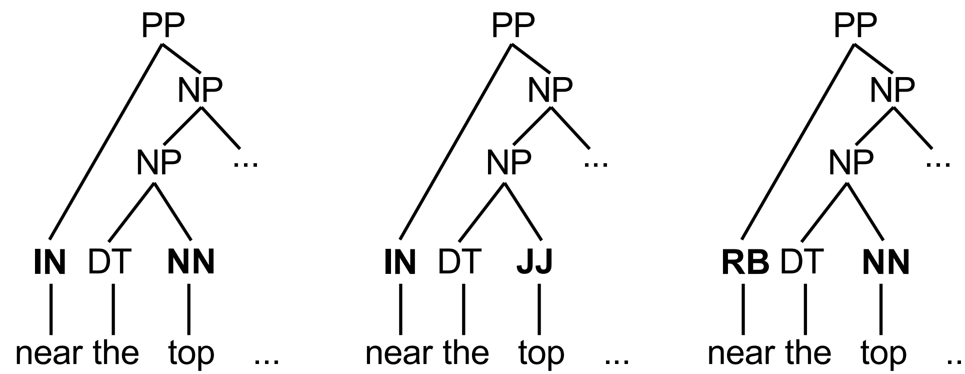
$\lambda_{There\ NNP\ NP\ NP\ PP\ RB\ near} = 0.2$



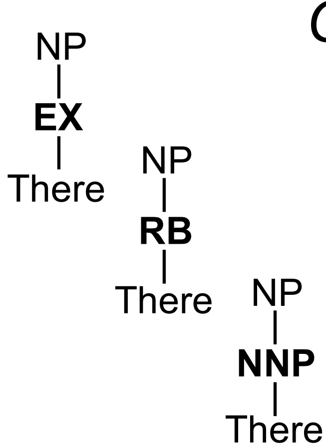


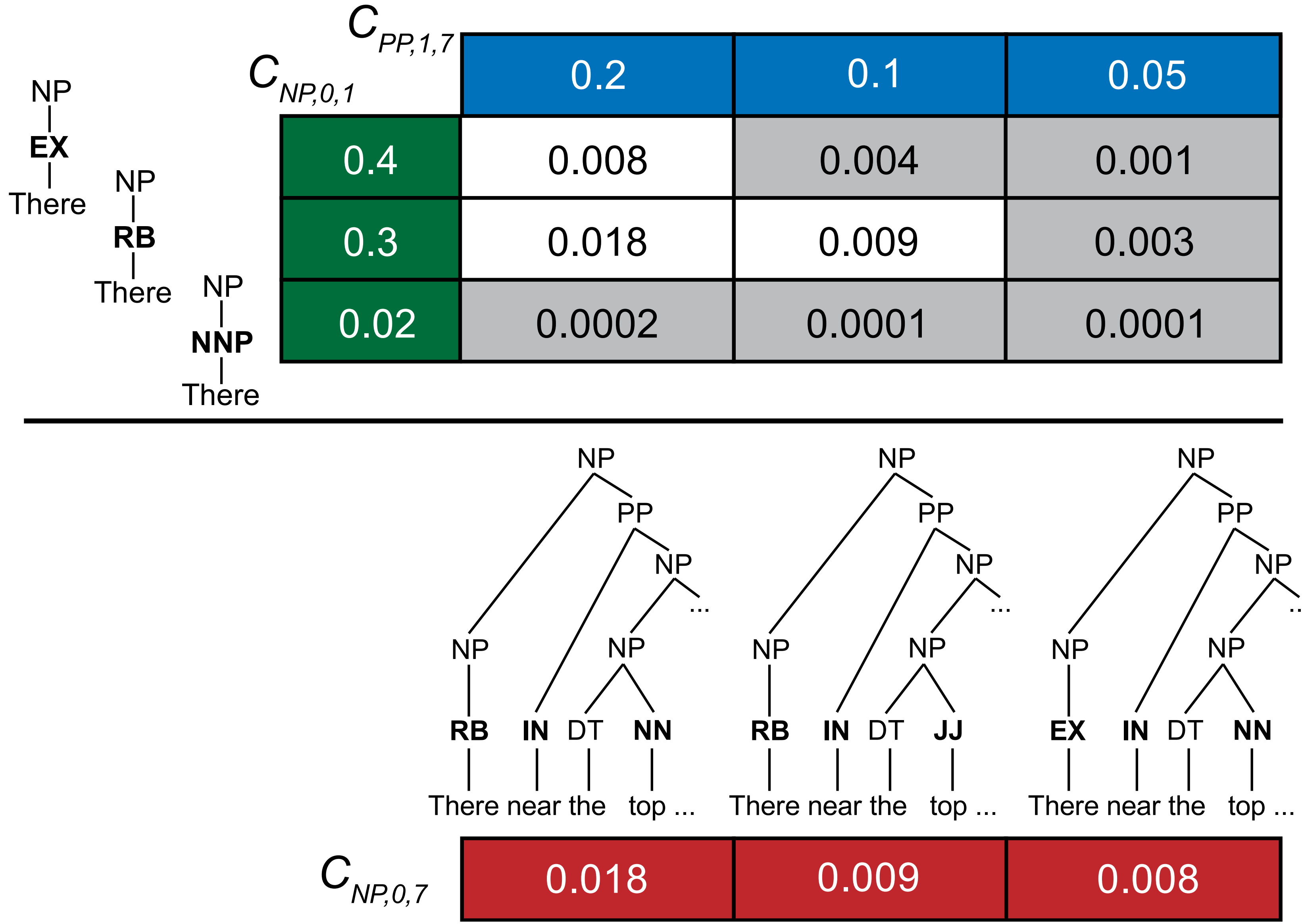
$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04 × 0.2	0.02 × 0.2	0.01 × 0.1
0.3	0.03 × 0.6	0.015 × 0.6	0.0075 × 0.4
0.02	0.002 × 0.1	0.001 × 0.1	0.0005 × 0.2


PP
NP
NP
...
IN DT NN
near the top ...
PP
NP
NP
...
IN DT JJ
near the top ...
PP
NP
NP
...
RB DT NN
near the top ...
$C_{PP,1,7}$
$C_{NP,0,1}$
0.2
0.1
0.05
NP
EX
There
0.4
0.008
0.004
0.001
NP
RB
There
0.3
0.018
0.009
0.003
NP
NNP
There
0.02
0.0002
0.0001
0.0001



$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.008	0.004	0.001
0.3	0.018	0.009	0.003
0.02	0.0002	0.0001	0.0001





$C_{PP,1,7}$
$C_{NP,0,1}$
0.2
0.1
0.05
0.4
0.008
0.004
0.001
0.3
0.018
0.009
0.003
0.02
0.0002
0.0001
0.0001
NP
EX
There
NP
RB
There
NP
NNP
There
NP
PP
NP
...
NP
NP
RB IN DT NN
There near the top ...
NP
PP
NP
...
NP
NP
RB IN DT JJ
There near the top ...
NP
PP
NP
...
NP
NP
EX IN DT NN
There near the top ...
$C_{NP,0,7}$
0.018
0.009
0.008

Clarification

- Cube pruning does not actually expand all k^2 proofs as this example showed

- It uses a fast approximation that only looks at $O(k)$ proofs