

# Probabilistic Context-free Grammars

Computational Linguistics

Alexander Koller

24 November 2023

# The CKY Recognizer

$S \rightarrow NP \ VP$   
 $NP \rightarrow Det \ N$   
 $VP \rightarrow V \ NP$

$V \rightarrow ate$   
 $NP \rightarrow John$

$Det \rightarrow a$   
 $N \rightarrow sandwich$

Chart

$S \Rightarrow^* w$

	$i = 1$	2	3	4
5	S	VP	NP	N
4			Det	...
3		V	...	
$k = 2$	NP	...		

John  
 ate  
 a  
 sandwich

Cell at column  $i$ , row  $k$ :  
 $\{ A \mid A \Rightarrow^* w_i \dots w_{k-1} \}$

# CKY recognizer: pseudocode

Data structure:  $\text{Ch}(i,k)$  eventually contains  $\{A \mid A \Rightarrow^* w_i \dots w_{k-1}\}$   
(initially all empty).

for each  $i$  from 1 to  $n$ :

  for each production rule  $A \rightarrow w_i$ :  
    add  $A$  to  $\text{Ch}(i, i+1)$

for each *width*  $b$  from 2 to  $n$ :

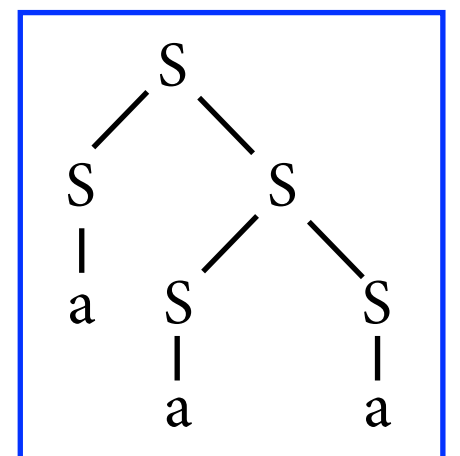
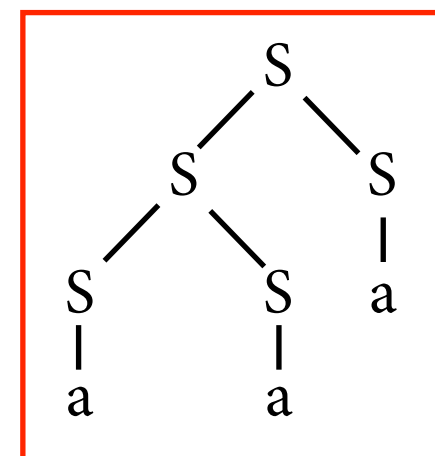
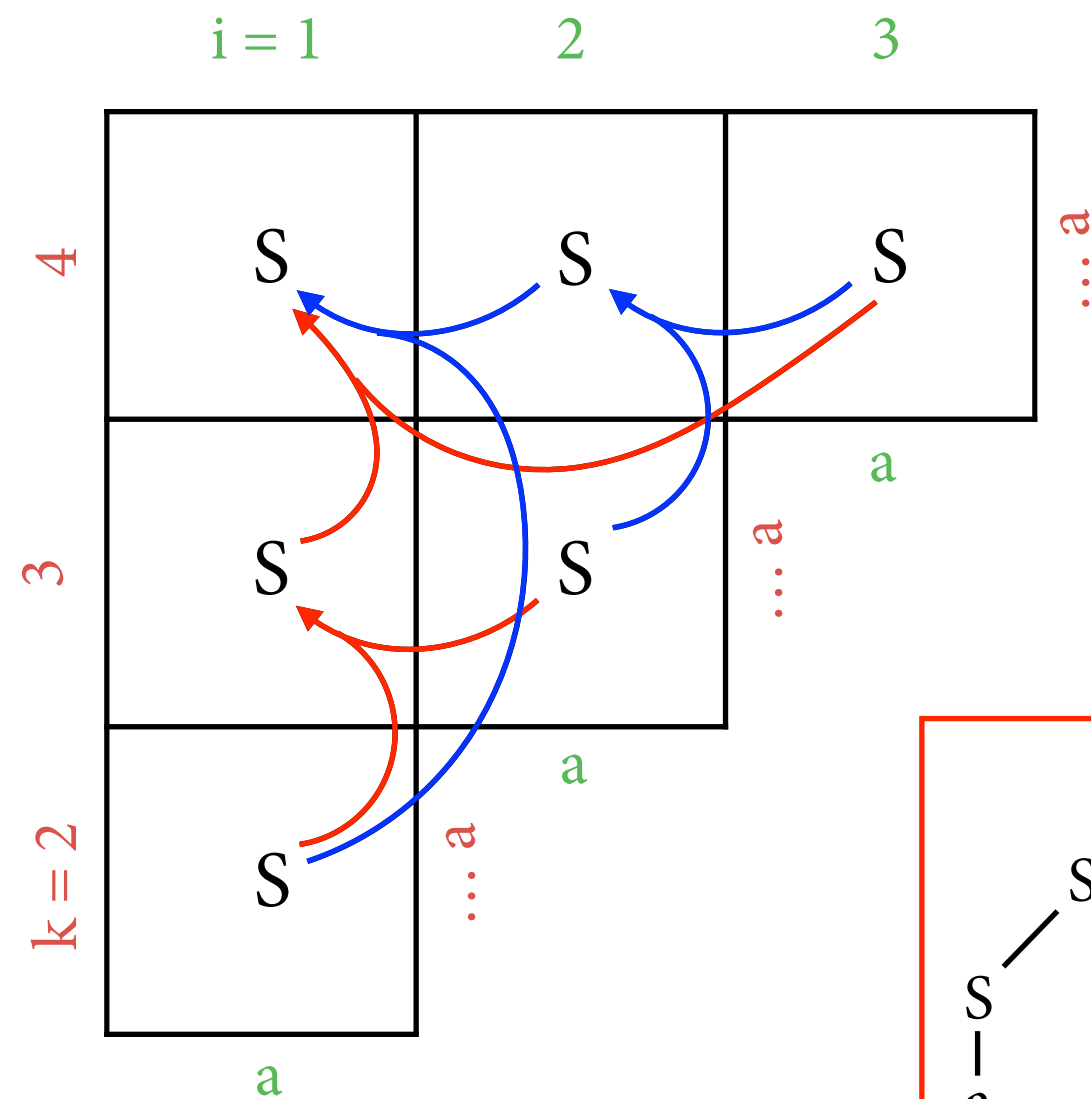
  for each *start position*  $i$  from 1 to  $n-b+1$ :  
    for each *left width*  $k$  from 1 to  $b-1$ :  
      for each  $B \in \text{Ch}(i, i+k)$  and  $C \in \text{Ch}(i+k, i+b)$ :  
        for each production rule  $A \rightarrow B C$ :  
          add  $A$  to  $\text{Ch}(i, i+b)$

claim that  $w \in L(G)$  iff  $S \in \text{Ch}(1, n+1)$

# Backpointers

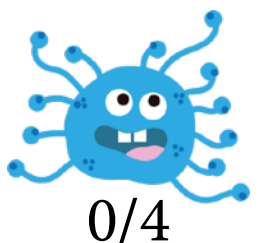
$S \rightarrow S S$

$S \rightarrow a$



# Recognizer to Parser

- Parser: need to construct parse trees from chart.
- Do this by memorizing how each  $A \in \text{Ch}(i,k)$  can be constructed from smaller parts.
  - ▶ built from  $B \in \text{Ch}(i,j)$  and  $C \in \text{Ch}(j,k)$  using  $A \rightarrow B C$ : store  $(B,C,j)$  in *backpointer* for  $A$  in  $\text{Ch}(i,k)$ .
  - ▶ analogous to backpointers in HMMs
- Once chart has been filled, enumerate trees recursively by following backpointers, starting at  $S \in \text{Ch}(1,n+1)$ .



# Outline

- Probabilistic context-free grammars.
- The Viterbi CKY algorithm.
- Getting a grammar from a treebank.
- Evaluating PCFG parsers.

# Let's play a game

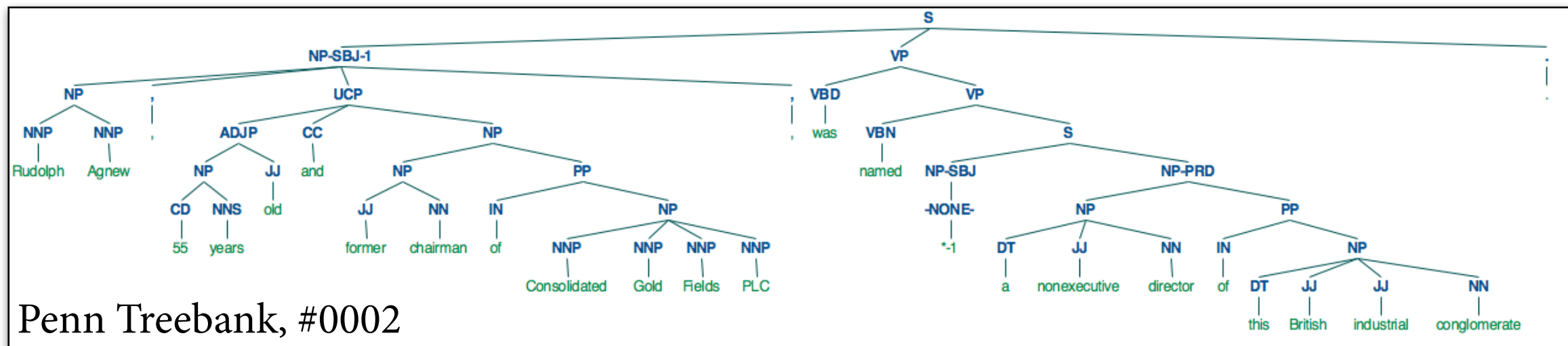
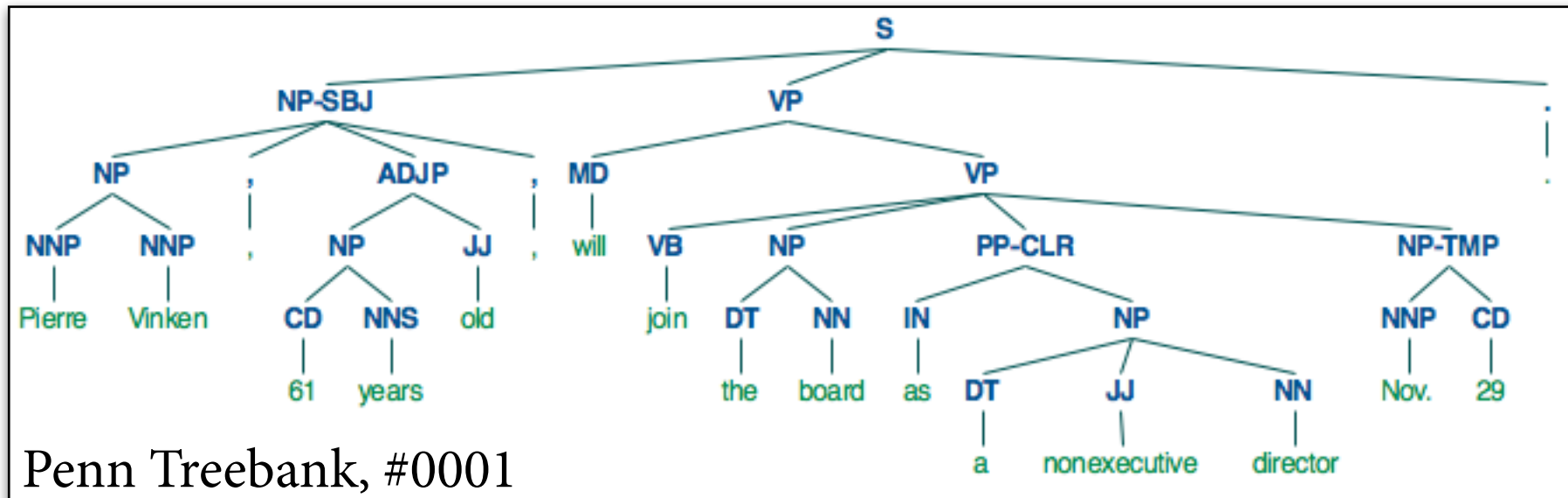
- Given a nonterminal symbol, expand it.
- You can take one of two moves:
  - ▶ expand nonterminal into a sequence of other nonterminals
  - ▶ use nonterminals S, NP, VP, PP, ... or POS tags
  - ▶ expand nonterminal into a word

# Penn Treebank POS tags

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	<i>and, but, or</i>	SYM	Symbol	<i>+, %, &amp;</i>
CD	Cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	Determiner	<i>a, the</i>	UH	Interjection	<i>ah, oops</i>
EX	Existential ‘there’	<i>there</i>	VB	Verb, base form	<i>eat</i>
FW	Foreign word	<i>mea culpa</i>	VBD	Verb, past tense	<i>ate</i>
IN	Preposition/sub-conj	<i>of, in, by</i>	VBG	Verb, gerund	<i>eating</i>
JJ	Adjective	<i>yellow</i>	VBN	Verb, past participle	<i>eaten</i>
JJR	Adj., comparative	<i>bigger</i>	VBP	Verb, non-3sg pres	<i>eat</i>
JJS	Adj., superlative	<i>wildest</i>	VBZ	Verb, 3sg pres	<i>eats</i>
LS	List item marker	<i>1, 2, One</i>	WDT	Wh-determiner	<i>which, that</i>
MD	Modal	<i>can, should</i>	WP	Wh-pronoun	<i>what, who</i>
NN	Noun, sing. or mass	<i>llama</i>	WP\$	Possessive wh-	<i>whose</i>
NNS	Noun, plural	<i>llamas</i>	WRB	Wh-adverb	<i>how, where</i>
NNP	Proper noun, singular	<i>IBM</i>	\$	Dollar sign	<i>\$</i>
NNPS	Proper noun, plural	<i>Carolinas</i>	#	Pound sign	<i>#</i>
PDT	Predeterminer	<i>all, both</i>	“	Left quote	<i>( ‘ or “)</i>
POS	Possessive ending	<i>’s</i>	”	Right quote	<i>( ’ or ”)</i>
PP	Personal pronoun	<i>I, you, he</i>	(	Left parenthesis	<i>( [ , { , &lt;)</i>
PP\$	Possessive pronoun	<i>your, one’s</i>	)	Right parenthesis	<i>( ] , } , &gt;)</i>
RB	Adverb	<i>quickly, never</i>	,	Comma	<i>,</i>
RBR	Adverb, comparative	<i>faster</i>	.	Sentence-final punc	<i>( . ! ?)</i>
RBS	Adverb, superlative	<i>fastest</i>	:	Mid-sentence punc	<i>( : ; ... – -)</i>
RP	Particle	<i>up, off</i>			



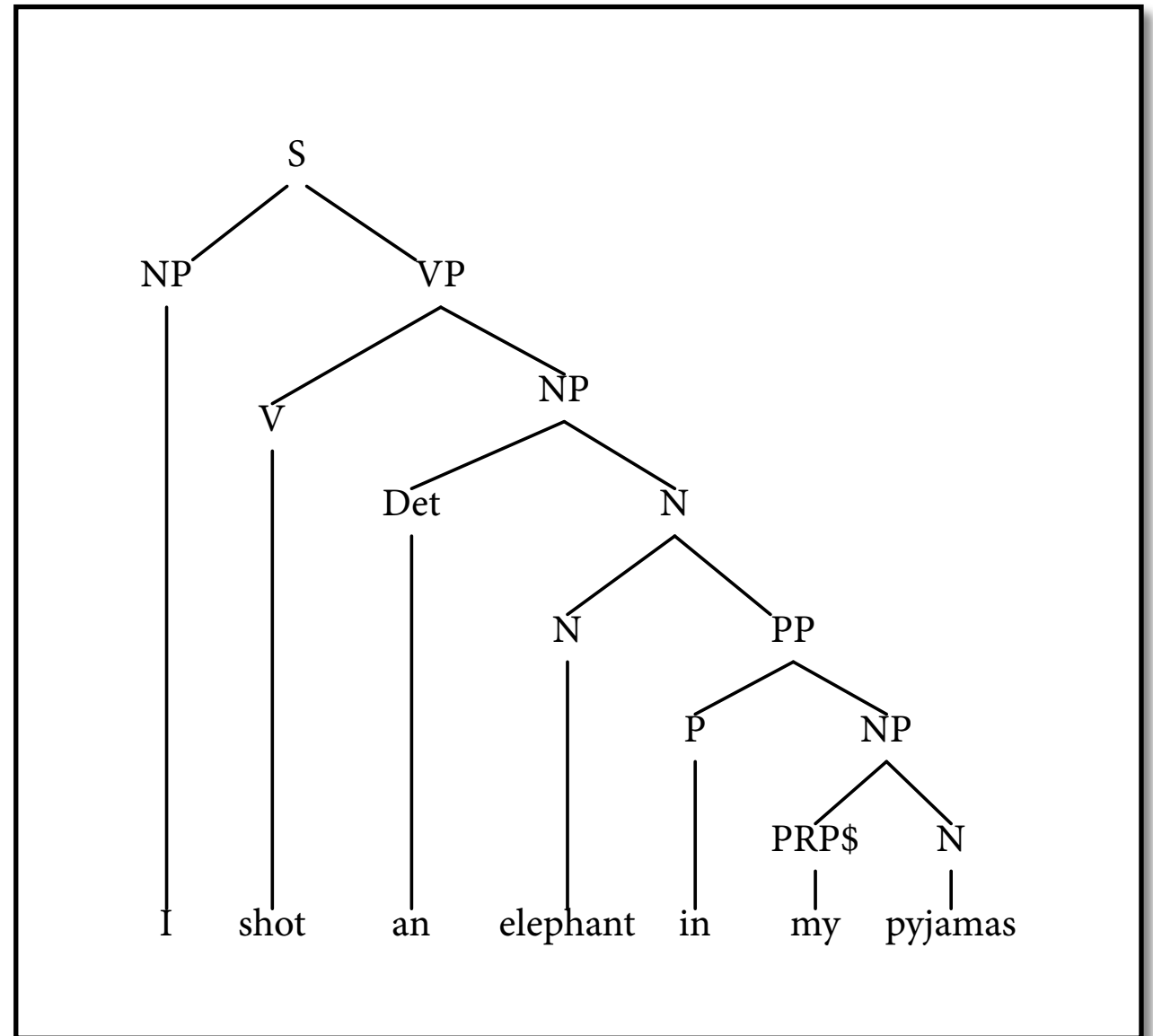
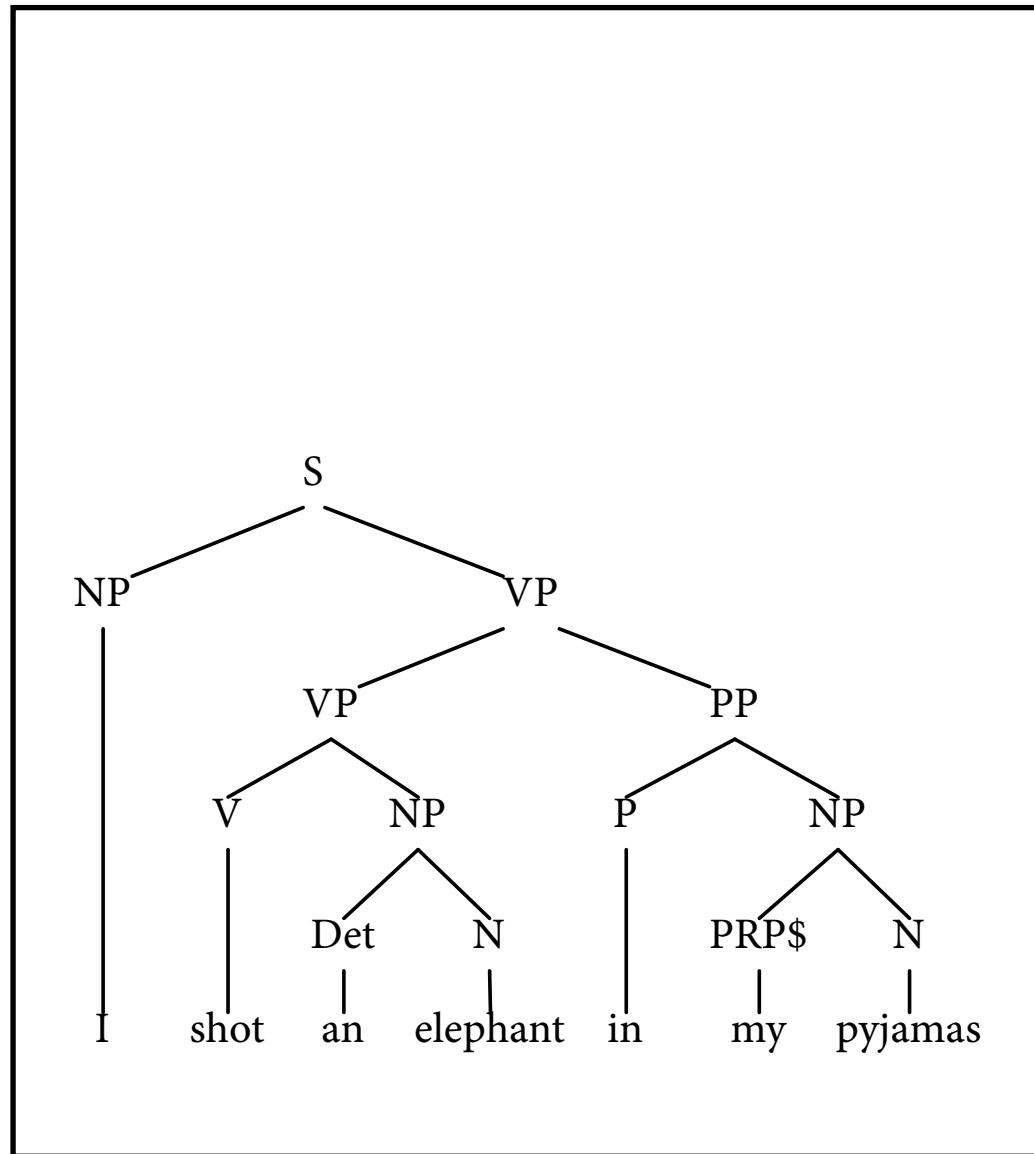
# Some real trees



`nltk.corpus.treebank.parsed_sents("wsj_0001.mrg")[0].draw()`

# Ambiguity

Need to *disambiguate*: find “correct” parse tree for ambiguous sentence.



How do we identify the “correct” tree?

How do we compute it efficiently? (Remember: exponential number of readings.)

# Probabilistic CFGs

- A *probabilistic context-free grammar (PCFG)* is a context-free grammar in which
  - ▶ each production rule  $A \rightarrow w$  has a probability  $P(A \rightarrow w \mid A)$ : when we expand  $A$ , how likely is it that we choose  $A \rightarrow w$ ?
  - ▶ for each nonterminal  $A$ , probabilities must sum to one:

$$\sum_w P(A \rightarrow w \mid A) = 1$$

- ▶ we will write  $P(A \rightarrow w)$  instead of  $P(A \rightarrow w \mid A)$  for short

# An example

$S \rightarrow NP \ VP$	[1.0]	$VP \rightarrow V \ NP$	[0.5]
$NP \rightarrow Det \ N$	[0.8]	$VP \rightarrow VP \ PP$	[0.5]
$NP \rightarrow i$	[0.2]	$V \rightarrow shot$	[1.0]
$N \rightarrow N \ PP$	[0.4]	$PP \rightarrow P \ NP$	[1.0]
$N \rightarrow elephant$	[0.3]	$P \rightarrow in$	[1.0]
$N \rightarrow pyjamas$	[0.3]	$Det \rightarrow an$	[0.5]
		$Det \rightarrow my$	[0.5]

(let's pretend for simplicity that Det = PRP\$)

# Generative process

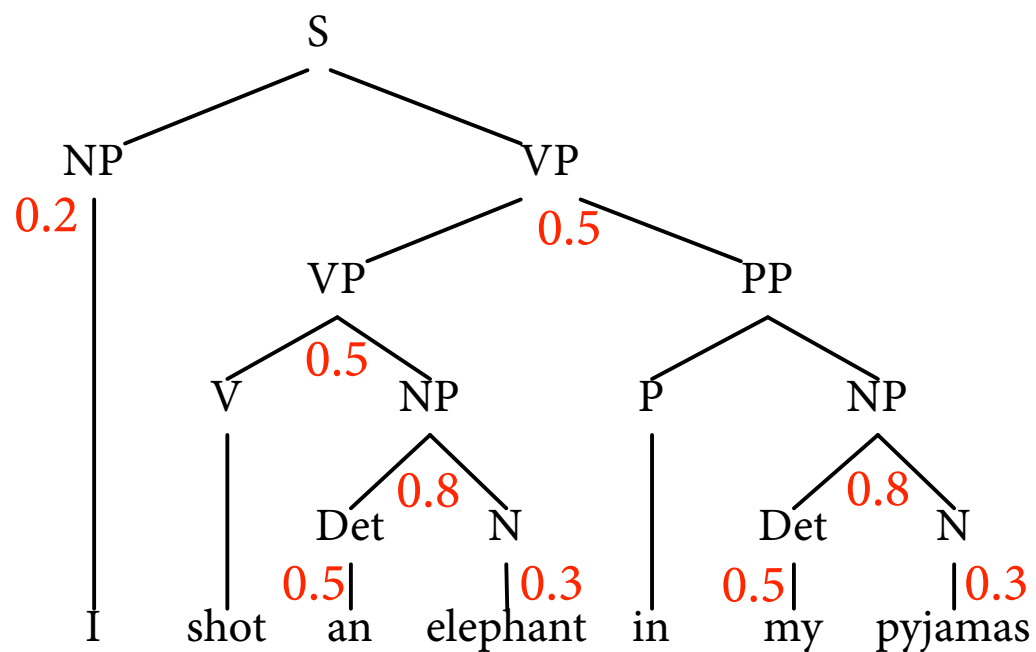
- PCFG generates random derivations of CFG.
  - ▶ each event (expand nonterminal by production rule) statistically independent of all the others

$S \xRightarrow{1.0} NP \ VP \xRightarrow{0.2} i \ VP \xRightarrow{0.5} i \ VP \ PP$   
 $\Rightarrow^* i \text{ shot an elephant in my pyjamas}$  0.00072

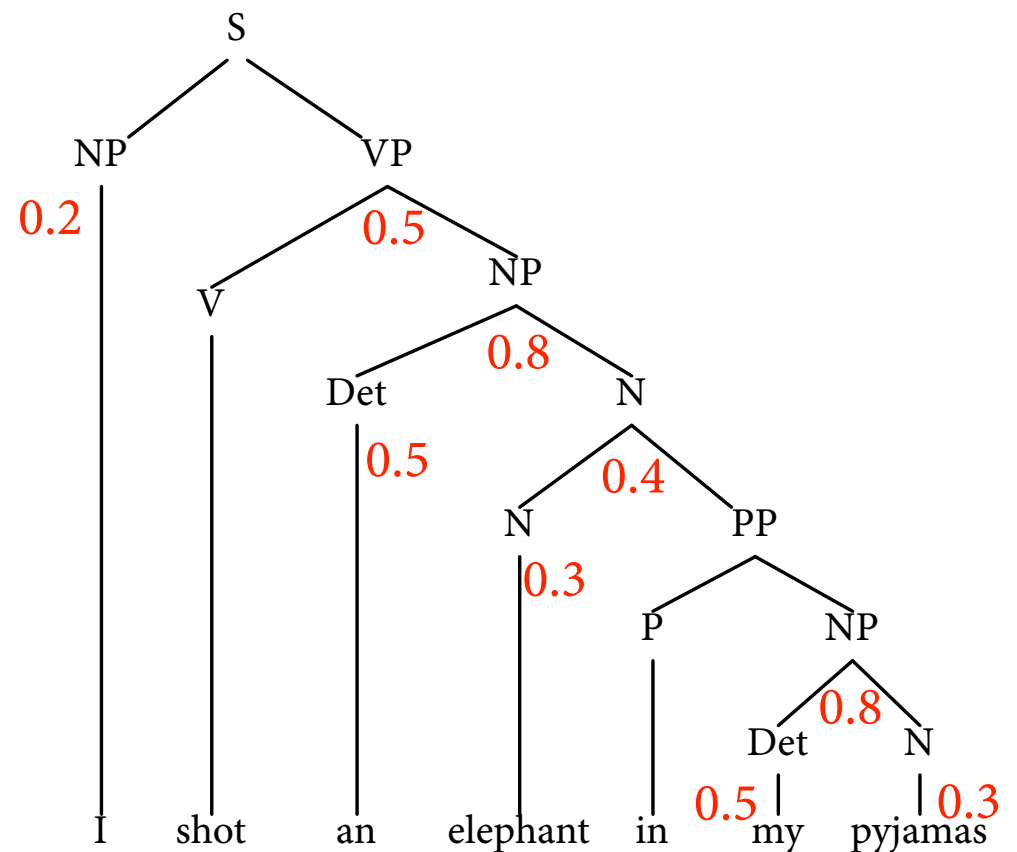
$S \xRightarrow{1.0} NP \ VP \xRightarrow{0.2} i \ VP \xRightarrow{0.4} i \ V \ Det \ N$   
 $\Rightarrow i \ V \ Det \ N \ PP \Rightarrow^* i \text{ shot ... pyjamas}$  0.00057  
0.4

# Parse trees

$$P(t_1) = 0.00072$$



$$P(t_2) = 0.00057$$



↑  
“correct” = more probable parse tree

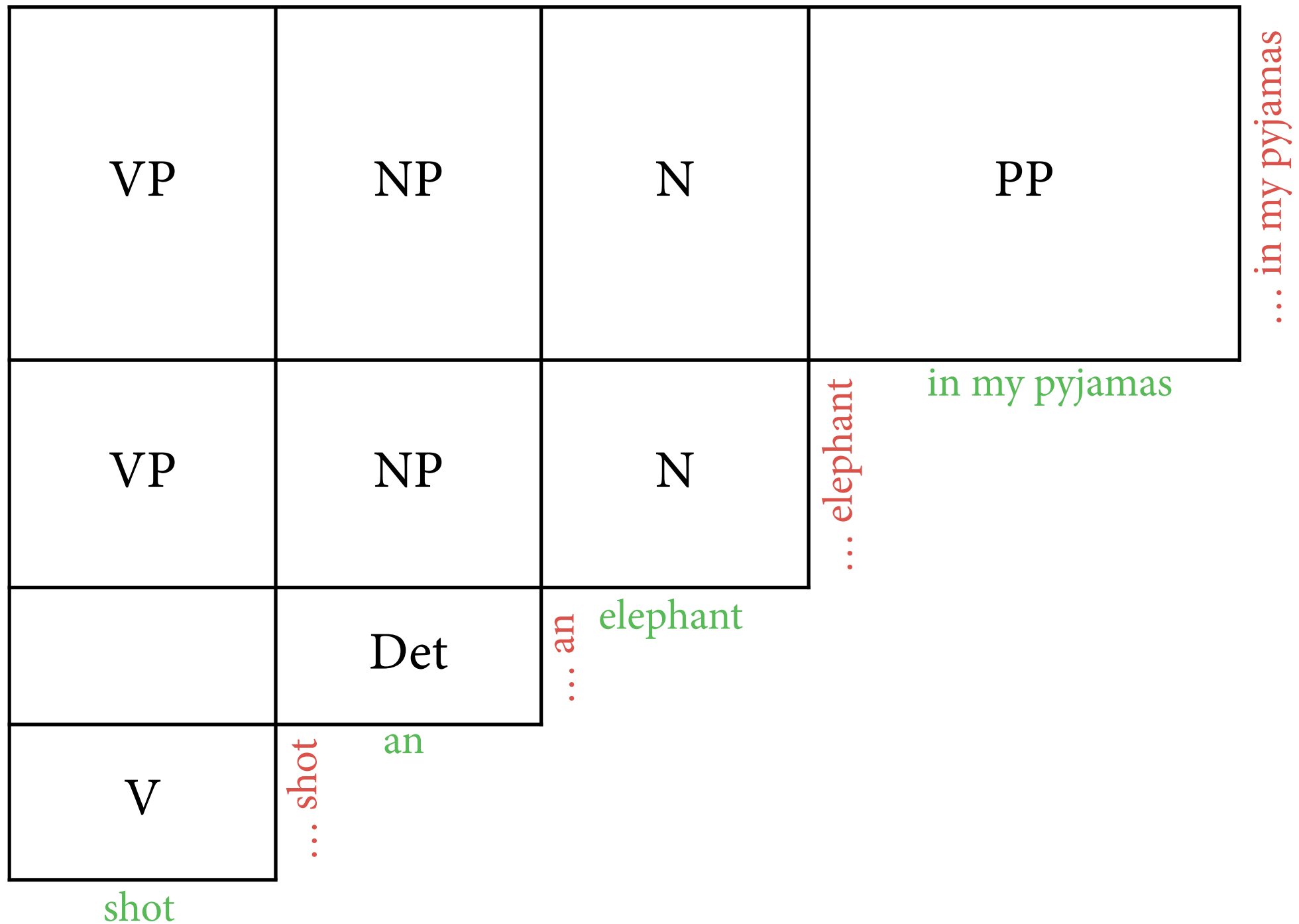


# Disambiguation

- Assumption: “correct” parse tree = the parse tree that had highest prob of being generated by random process, i.e.  $\operatorname{argmax}_{t \in \text{pares}(w)} P(t)$
- We use a variant of the Viterbi algorithm to compute it.
- Here, Viterbi based on CKY; can do it with other parsing algorithms too.

# The intuition

Ordinary CKY parse chart:  $\text{Ch}(i,k) = \{A \mid A \Rightarrow^* w_i \dots w_{k-1}\}$



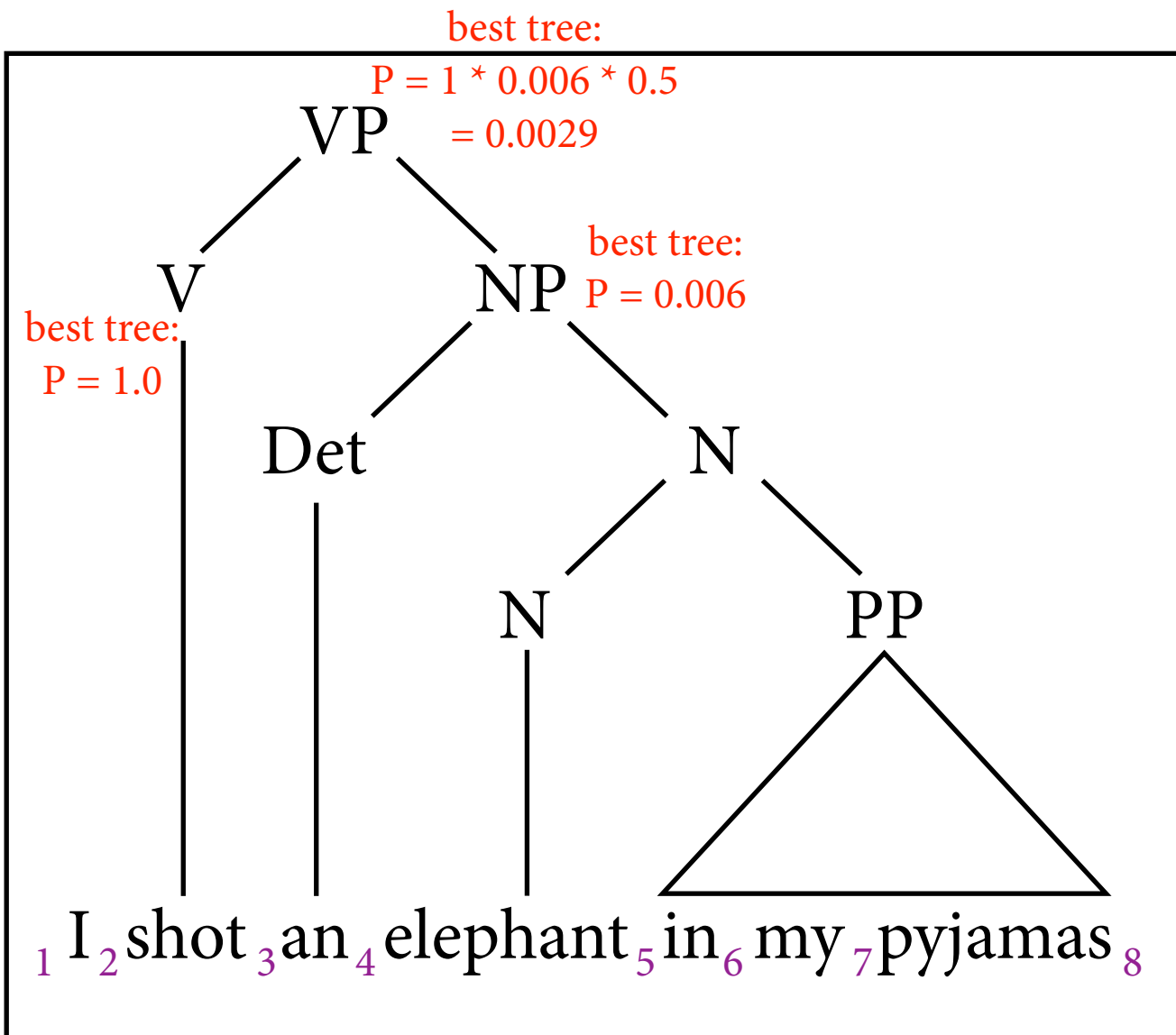


# The intuition

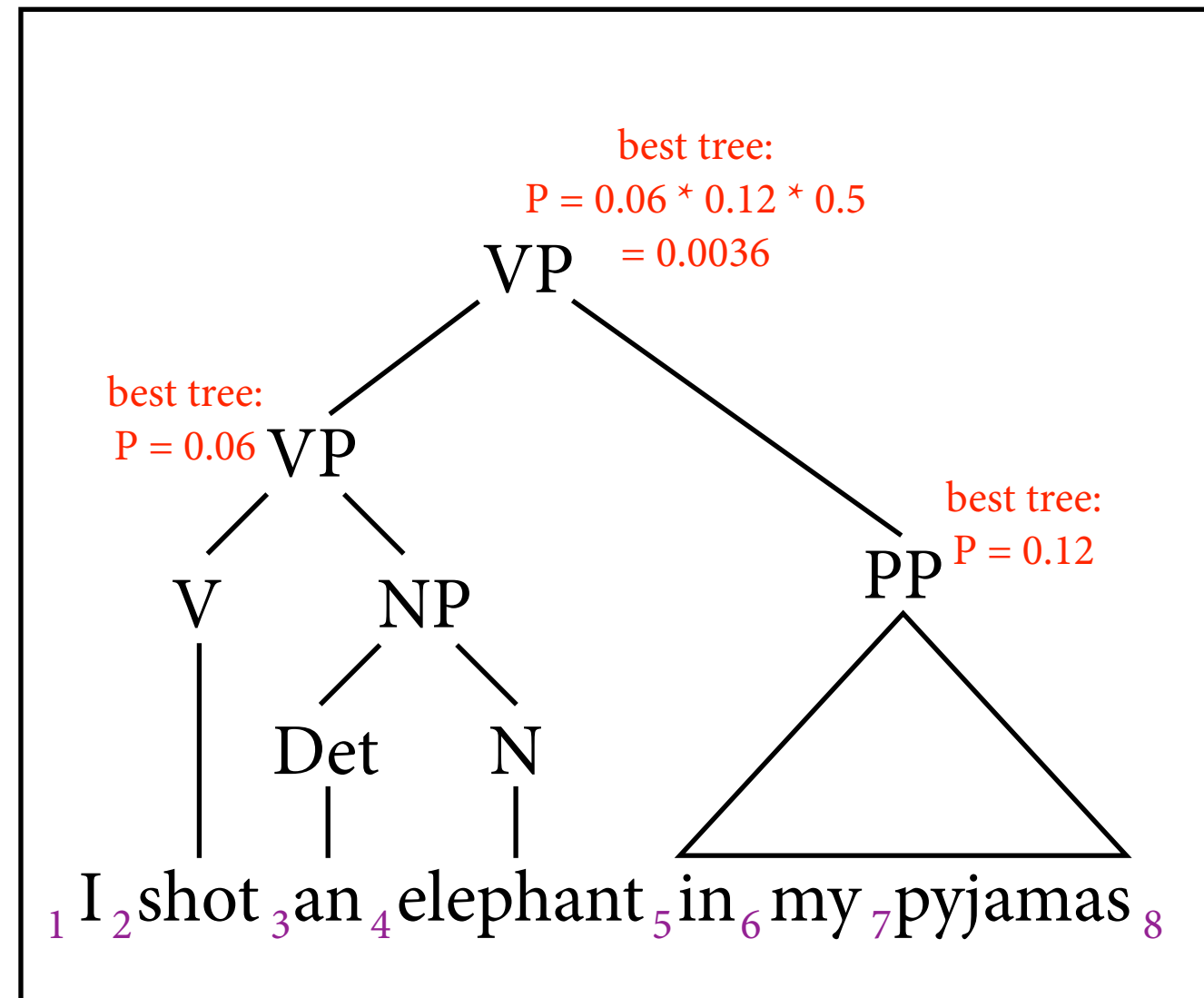
Viterbi CKY parse chart:  $\text{Ch}(i, k) = \{(A, p) \mid p = \max_{d: A \Rightarrow^* w_i \dots w_{k-1}} P(d)\}$

VP: 0.0036	NP: 0.006	N: 0.014	PP: 0.12	... in my pyjamas
VP: 0.06	NP: 0.12	N: 0.3	... elephant	in my pyjamas
	Det: 0.5	... an	elephant	
V: 1.0	... shot	an		
shot				

# The intuition



split 2-8 into 2-3, 3-8  
expand  $VP \rightarrow V NP$



split 2-8 into 2-5, 5-8  
expand  $VP \rightarrow VP PP$

# Viterbi CKY

- Define for each span (i,k) and each nonterminal A the probability

$$V(A, i, k) = \max_{A \xRightarrow{d} w_i \dots w_{k-1}} P(d)$$

- Compute V iteratively “bottom up”, i.e. starting from small spans and working our way up to longer spans.

$$V(A, i, i + 1) = P(A \rightarrow w_i)$$

$$V(A, i, k) = \max_{\substack{A \rightarrow B \ C \\ i < j < k}} P(A \rightarrow B \ C) \cdot V(B, i, j) \cdot V(C, j, k)$$

# Viterbi CKY - pseudocode

set all  $V[A, i, j]$  to 0

for all  $i$  from 1 to  $n$ :

for all  $A$  with rule  $A \rightarrow w_i$ :

add  $A$  to  $Ch(i, i+1)$

$V[A, i, i+1] = P(A \rightarrow w_i)$

for all  $b$  from 2 to  $n$ :

for all  $i$  from 1 to  $n-b+1$ :

for all  $k$  from 1 to  $b-1$ :

for all  $B$  in  $Ch(i, i+k)$  and  $C$  in  $Ch(i+k, i+b)$ :

for all production rules  $A \rightarrow B C$ :

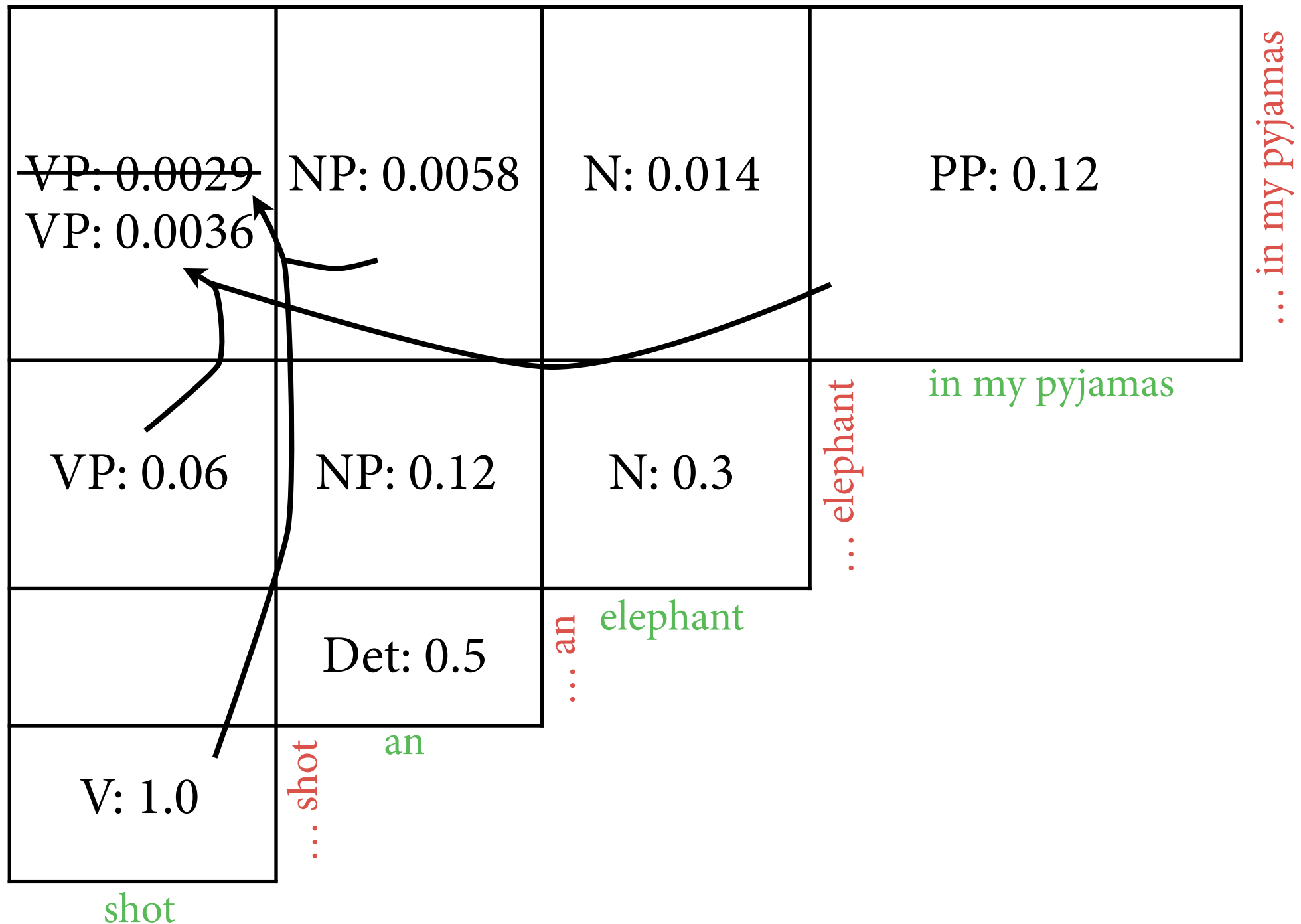
add  $A$  to  $Ch(i, i+b)$

if  $P(A \rightarrow B C) * V[B, i, i+k] * V[C, i+k, i+b] > V[A, i, i+b]$ :

$V[A, i, i+b] = P(A \rightarrow B C) * V[B, i, i+k] * V[C, i+k, i+b]$

# Viterbi-CKY in action

Viterbi CKY parse chart:  $\text{Ch}(i, k) = \{(A, p) \mid p = \max_{d: A \Rightarrow^* w_i \dots w_{k-1}} P(d)\}$



# Remarks

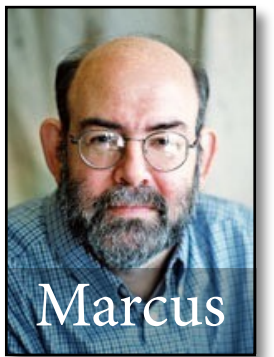
- Viterbi CKY has exactly the same nested loops as the ordinary CKY parser.
  - ▶ computing  $V$  in addition to  $Ch$  only changes constant factor
  - ▶ thus asymptotic runtime remains  $O(n^3)$
- Compute optimal parse by storing backpointers.
  - ▶ same backpointers as in ordinary CKY
  - ▶ sufficient to store the *best* backpointer for each  $(A,i,k)$  if we only care about best parse (and not all parses), i.e. actually uses less memory than ordinary CKY



# Obtaining the PCFG

- How to obtain the CFG?
  - ▶ write by hand
  - ▶ derive from *treebank*
  - ▶ *grammar induction* from raw text
- How to obtain the rule probabilities once we have the CFG?
  - ▶ maximum likelihood estimation from treebank
  - ▶ EM training from raw text (inside-outside algorithm)

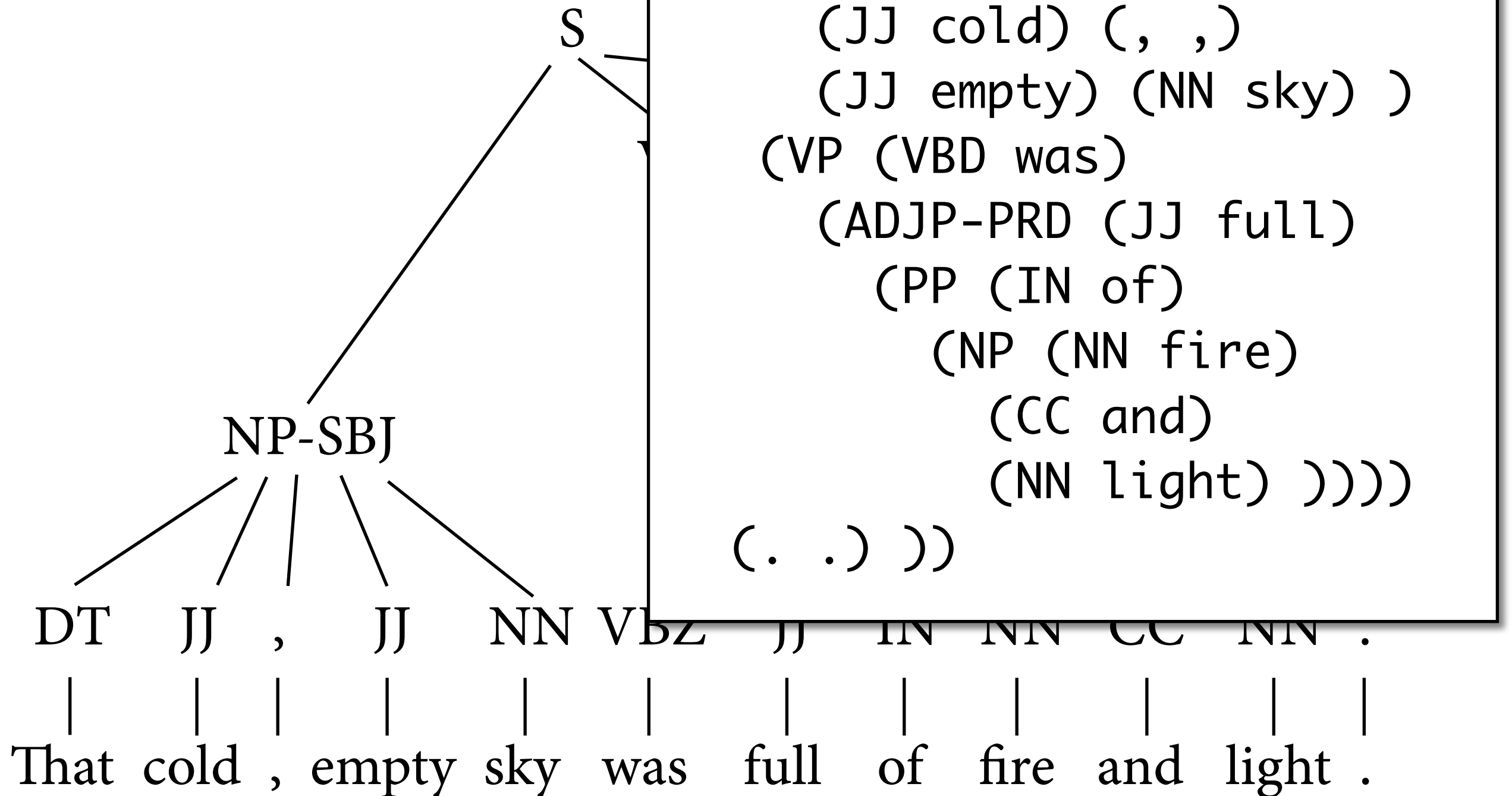
# The Penn Treebank



- Large (in the mid-90s) quantity of text, annotated with POS tags and syntactic structures.
- Consists of several sub-corpora:
  - ▶ Wall Street Journal: 1 year of news text, 1 million words
  - ▶ Brown corpus: balanced corpus, 1 million words
  - ▶ ATIS: dialogues on flight bookings, 5000 words
  - ▶ Switchboard: spoken dialogue, 3 million words
- WSJ PTB is standard corpus for training and evaluating PCFG parsers.



# Annotation format



# Reading off grammar

- Can directly read off “grammar in annotators’ heads” from trees in treebank.

- Yields very large CFG, e.g. 4500 rules for VP:

VP → VBD PP

VP → VBD PP PP

VP → VBD PP PP

VP → VBD PP PP

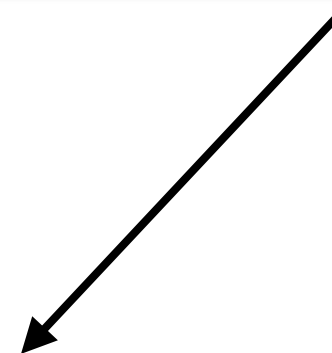
VP → VBD ADVP PP

VP → VBD PP ADVP

...

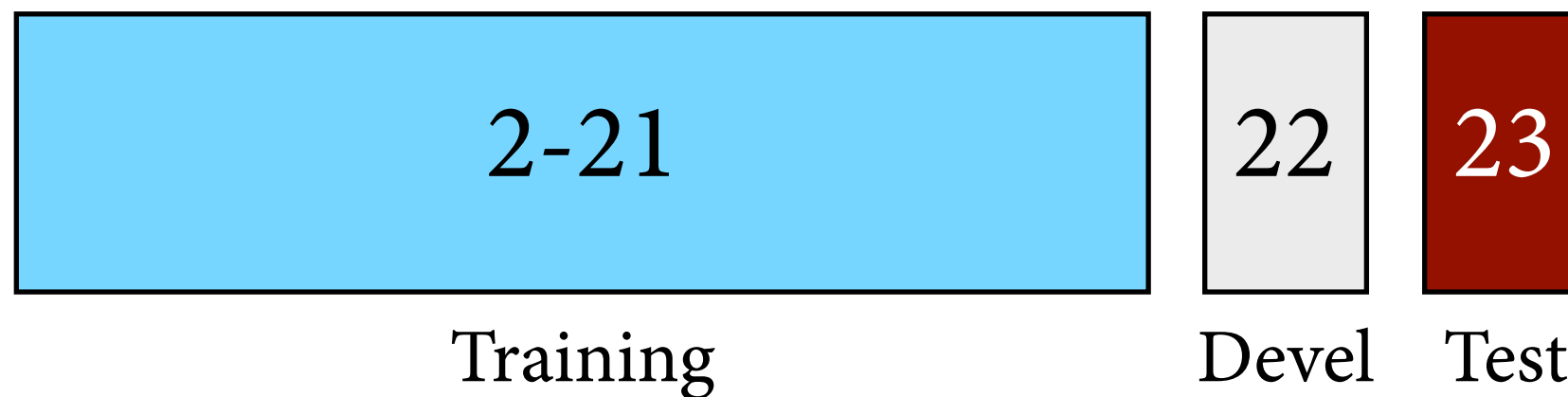
VP → VBD PP PP PP PP PP PP ADVP PP

“This mostly happens because we go  
from football in the fall to lifting in the winter  
to football again in the spring.”



# Evaluation

- Step 1: Decide on training and test corpus.  
For WSJ corpus, there is a conventional split by sections:

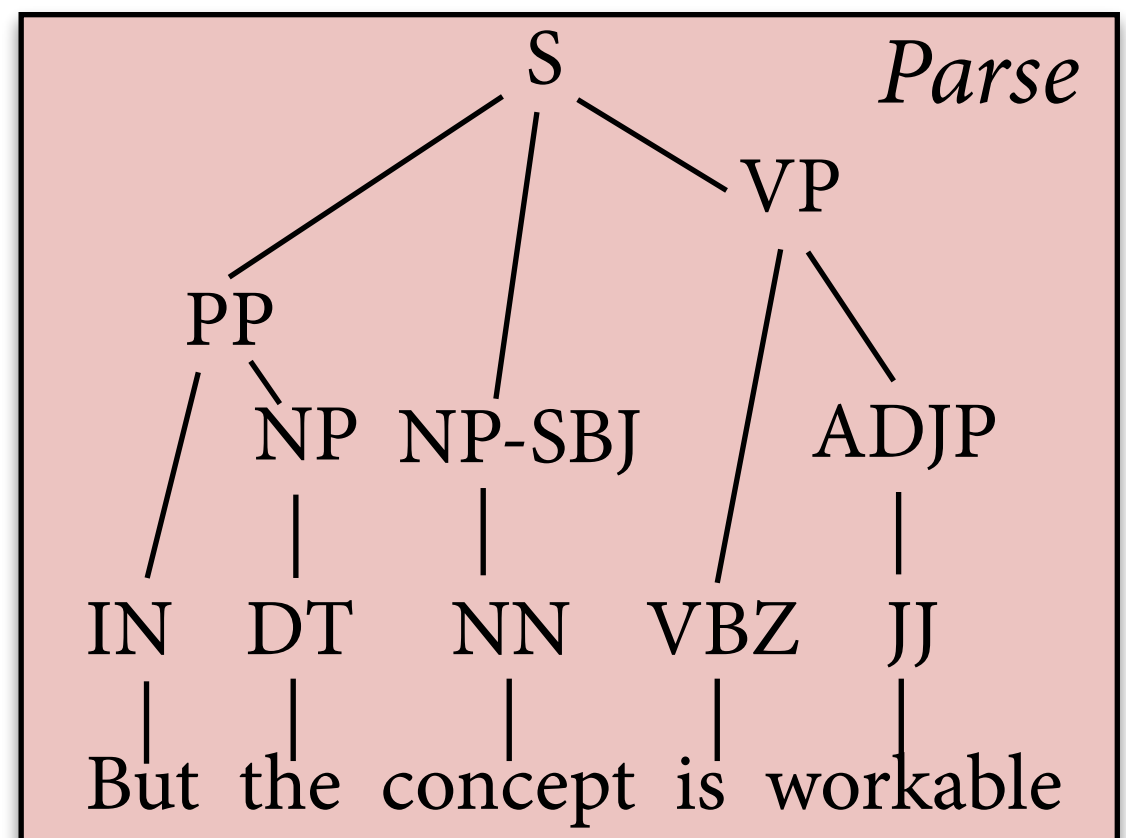
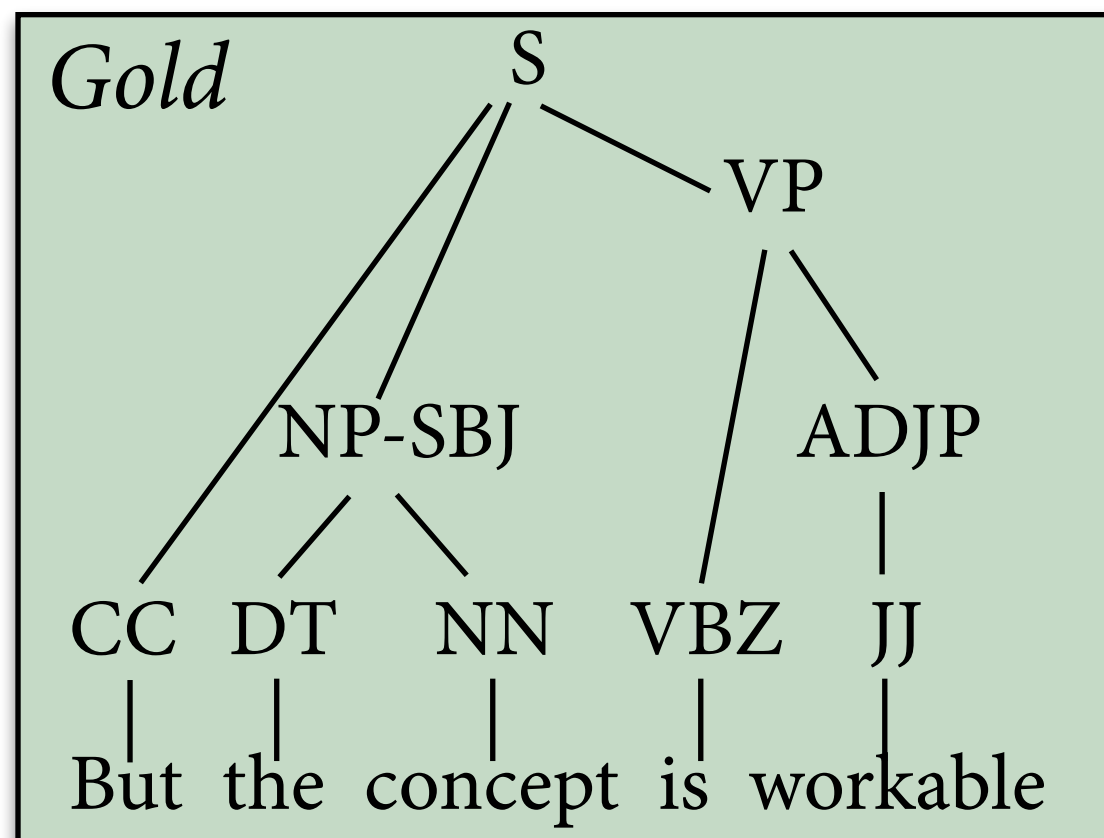


# Evaluation

- Step 2: How should we measure the accuracy of the parser?
- Straightforward idea: Measure “exact match”, i.e. proportion of gold standard trees that parser got right.
- This is too strict:
  - ▶ parser makes many decisions in parsing a sentence
  - ▶ a single incorrect parsing decision makes tree “wrong”
  - ▶ want more fine-grained measure

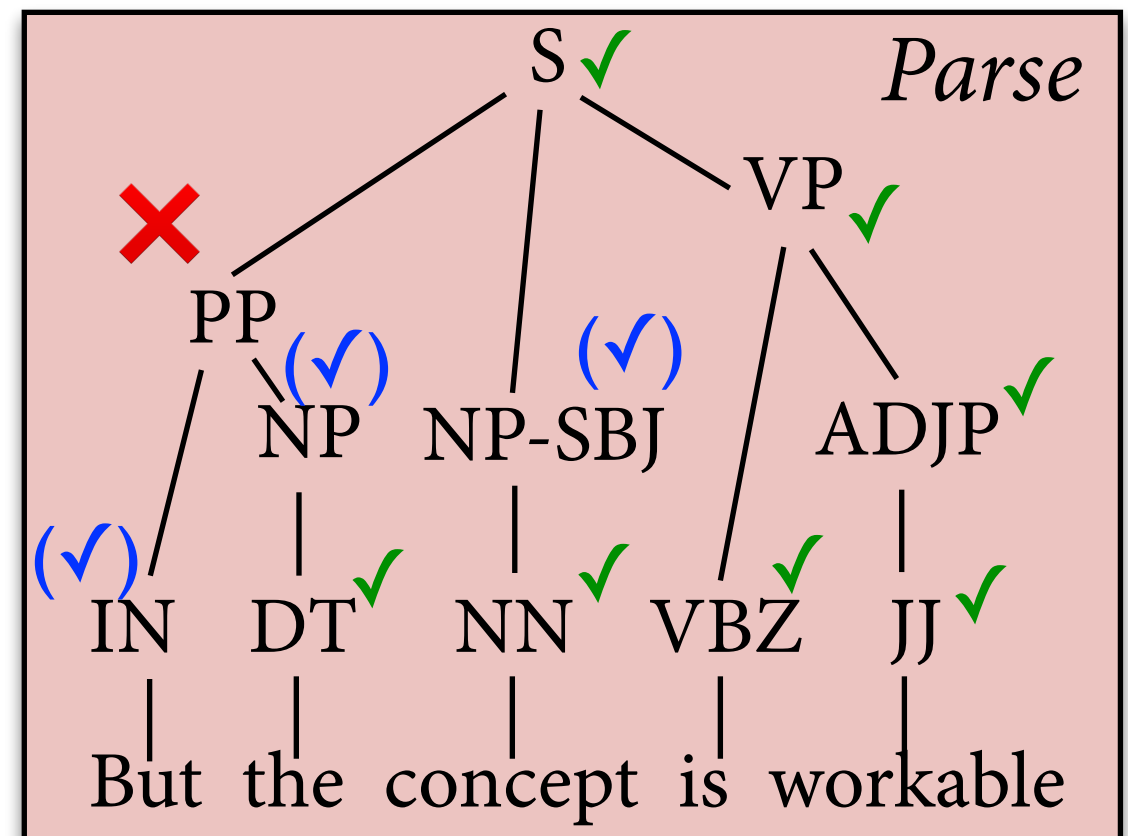
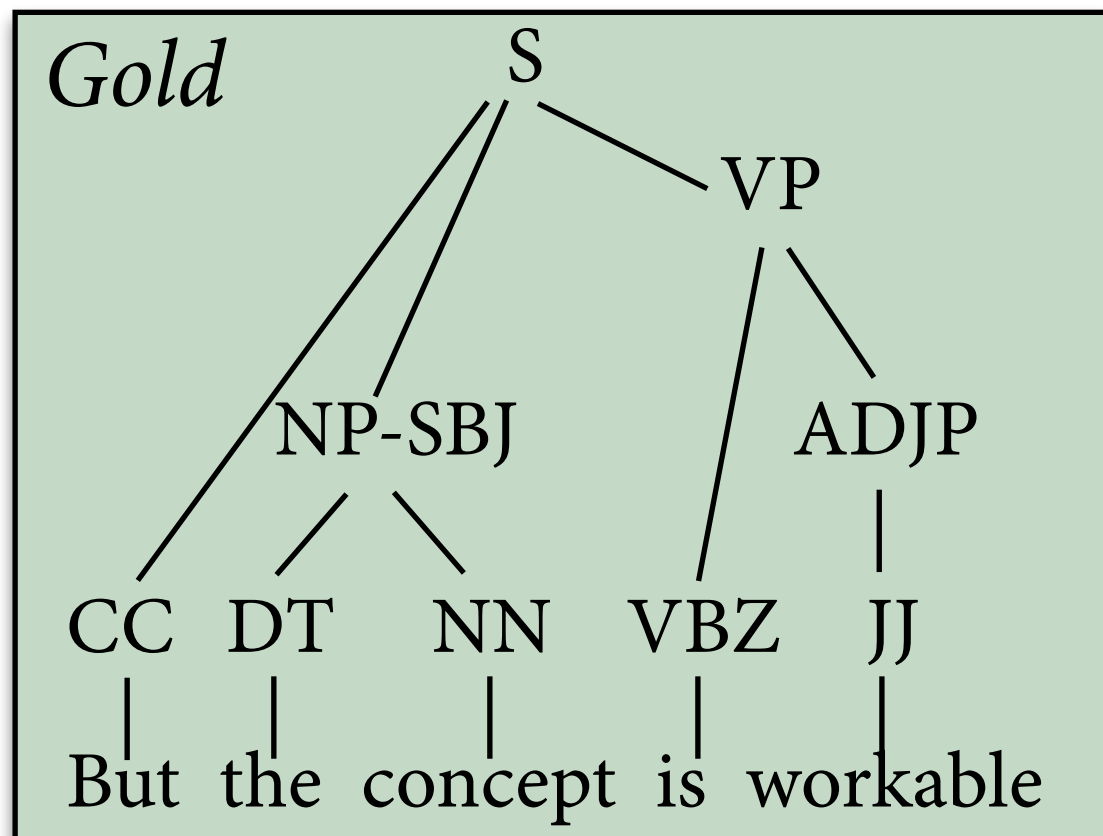
# Comparing parse trees

- Idea 2 (PARSEVAL): Compare *structure* of parse tree and gold standard tree.
  - ▶ Labeled: Which *constituents* (span + syntactic category) of one tree also occur in the other?
  - ▶ Unlabeled: How do the trees bracket the *substrings* of the sentence (ignoring syntactic categories)?



# Precision

What proportion of constituents in *parse tree* is also present in *gold tree*?

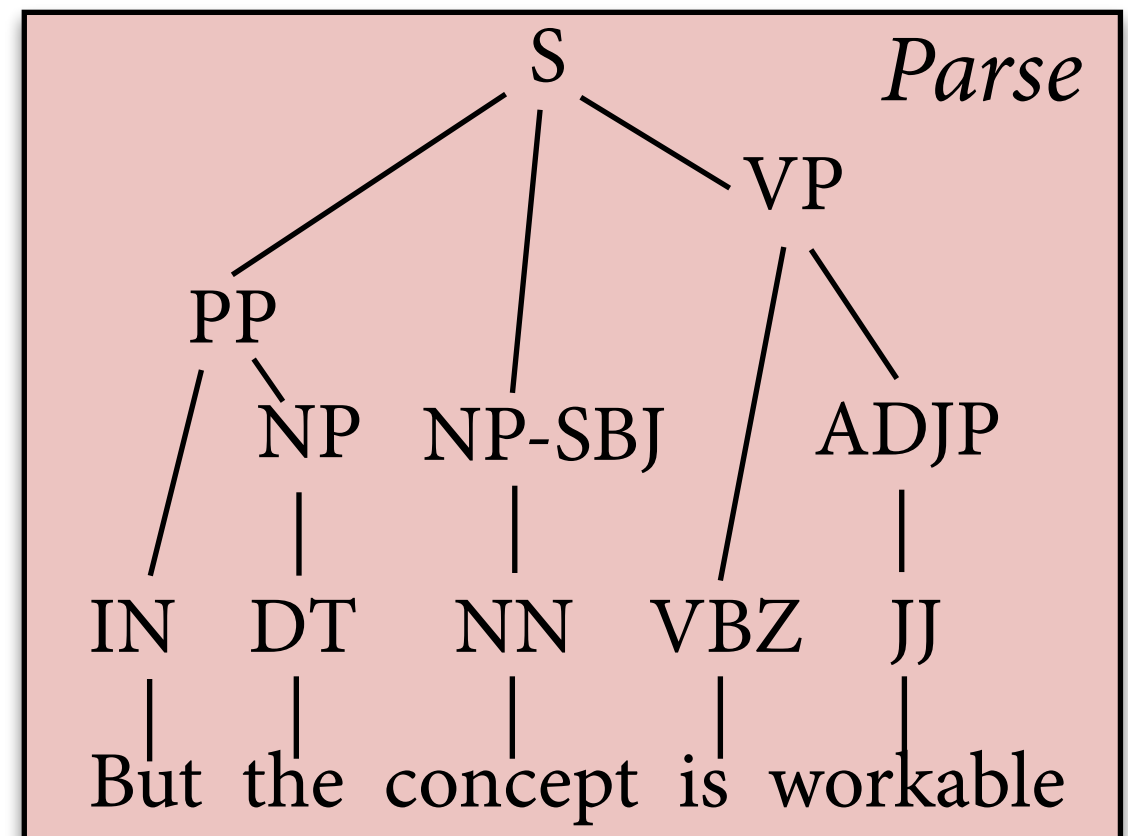
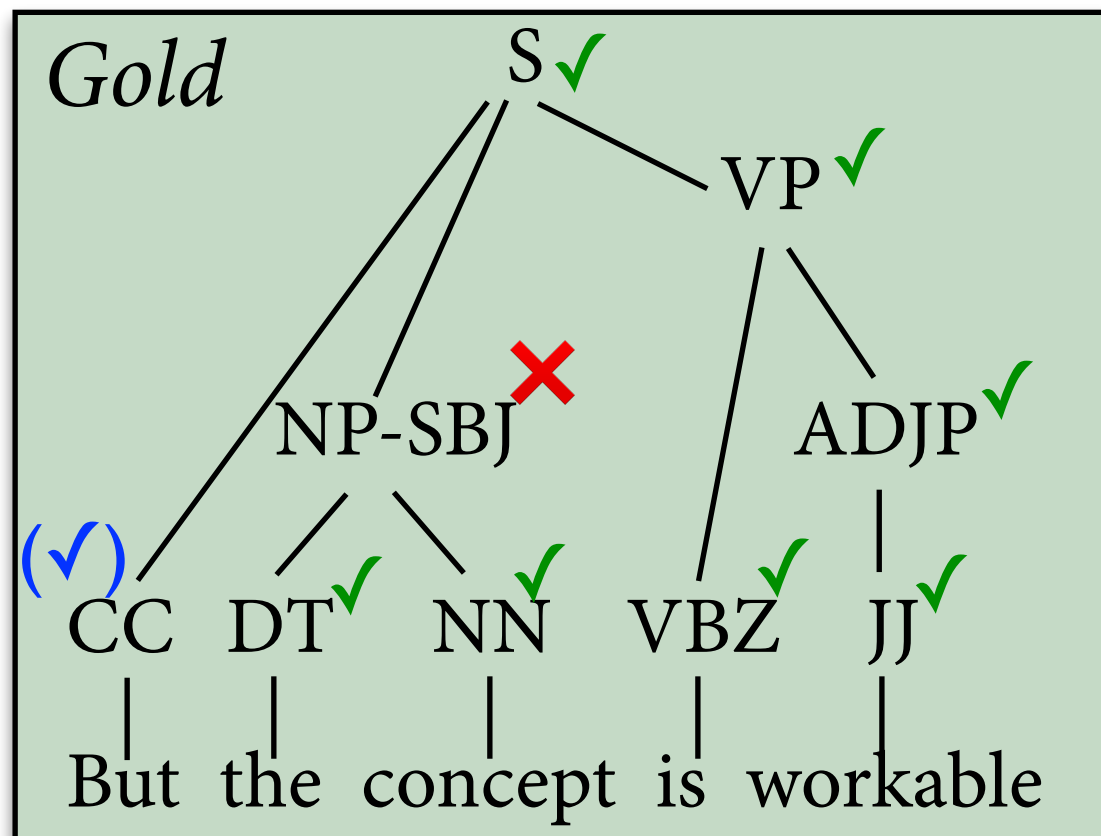


Labeled Precision =  $7 / 11 = 63.6\%$

Unlabeled Precision =  $10 / 11 = 90.9\%$

# Recall

What proportion of constituents in *gold tree* is also present in *parse tree*?



Labeled Recall =  $7 / 9 = 77.8\%$

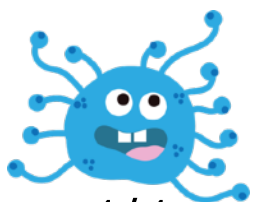
Unlabeled Recall =  $8 / 9 = 88.9\%$

# F-Score

- Precision and recall measure opposing qualities of a parser (“soundness” and “completeness”)
- Summarize both together in the *f-score*:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- In the example, we have labeled f-score 70.0 and unlabeled f-score 89.9.





# Summary

- PCFGs extend CFGs with rule probabilities.
  - ▶ Events of random process are nonterminal expansion steps. These are all statistically independent.
  - ▶ Use Viterbi CKY parser to find most probable parse tree for a sentence in cubic time.
- Read grammars off treebanks.
  - ▶ next time: learn rule probabilities
- Evaluation of statistical parsers.